UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Thiago Leucz Astrizi

Postquantum Preimage Chameleon Hash Functions for Digital Signatures

Florianópolis
2021

Thiago Leucz Astrizi

Postquantum Preimage Chameleon Hash Functions for Digital Signatures

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do título de mestre em Ciência da Computação.
Orientador: Prof. Ricardo Custódio, Dr.
Coorientadora: Profa. Lucia Moura, Dra.

Florianópolis
2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Astrizi, Thiago
Postquantum Preimage Chameleon Hash Functions for Digital Signatures / Thiago Astrizi ; orientador, Ricardo Custódio, coorientadora, Lucia Moura, 2021.
187 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Ciência da Computação, Florianópolis, 2021.

Inclui referências.

1. Ciência da Computação. 2. Hash Camaleão. 3. Assinaturas Digitais. 4. Provas de Segurança. I. Custódio, Ricardo. II. Moura, Lucia. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

Thiago Leucz Astrizi

Postquantum Preimage Chameleon Hash Functions for Digital Signatures

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Rafael de Santiago, Dr.
Universidade Federal de Santa Catarina

Prof. Edoardo Persichetti, Dr.
Florida Atlantic University

Prof. Julio César López Hernández, Dr.
Universidade Estadual de Campinas

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre em Ciência da Computação.

Documento assinado digitalmente
Vania Bogorny
Data: 24/06/2021 14:52:40-0300
CPF: 684.586.570-15
Verifique as assinaturas em https://v.ufsc.br

Coordenação do Programa de
Pós-Graduação

Documento assinado digitalmente
Ricardo Felipe Custodio
Data: 24/06/2021 16:00:37-0300
CPF: 485.981.609-97
Verifique as assinaturas em https://v.ufsc.br

Prof. Ricardo Custódio, Dr.
Orientador

Florianópolis, 2021.

AGRADECIMENTOS

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior- Brasil (CAPES) - Código de Financiamento 001.

Agradeço ao Professor Ricardo Custódio e à Professora Lucia Moura por terem feito parte desta pesquisa, pela orientação e disponibilidade para reuniões e discussões.

Agradeço aos vários colegas do Laboratório de Segurança em Computação da Universidade Federal de Santa Catarina (LabSEC-UFSC) que contribuíram com dicas e conselhos.

Agradeço à Lucila Bethânia de Souza Alosilla e Simone Cruz por terem ajudado resolvendo diferentes questões burocráticas na universidade e dando aconselhamentos.

Agradeço à minha tia Ana Leuch Lozovei por todo o apoio durante o mestrado.

Agradeço aos professores Rafael de Santiago, Edoardo Persichetti e Julio César López Hernández pelas suas sugestões e contribuições como membros da banca de defesa da dissertação.

ACKNOWLEDGEMENTS

This work was financed by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior- Brasil (CAPES) – Finance Code 001.

I thank Professor Ricardo Custódio and Professor Lucia Moura for being part of the development of this research, for the supervision and availability in meetings and discussions.

I thank multiple co-workers in computer security laboratory (LabSEC) in Federal University of Santa Catarina (UFSC) who contributed with tips and advices.

I thank Lucila Bethânia de Souza Alosilla and Simone Cruz for helping to solve bureaucratic problems in the university and for advices.

I thank my aunt Ana Leuch Lozovei for the help and support during the masters course.

I thank Professors Rafael de Santiago, Edoardo Persichetti and Julio César López Hernández for numerous suggestions and contributions while being part of the defense board of this dissertation.

RESUMO

Funções de hash camaleão são funções hash (ou funções de resumo criptográfico) com um segredo associado (seu "*trapdoor*") que permite o cálculo de sua primeira ou segunda pré-imagem somente para aqueles que o conhecem. Essa dissertação apresenta uma revisão sobre diferentes construções de hash camaleão e suas aplicações, em especial para o uso de assinaturas digitais. Também propomos uma nova construção de assinatura com uma prova de segurança no modelo do oráculo aleatório, adaptável para adversários clássicos e pós-quânticos. Apresentamos também resultados de sua implementação. A assinatura proposta tem uma propriedade nova de que um par formado pela mensagem e sua assinatura armazena qualquer dado arbitrário que seja escolhido durante a geração de chaves, o qual pode ser revelado computando um hash da mensagem e sua assinatura. Isso pode ser usado para construir assinaturas digitais mais amigáveis ao usuário e ligar um par de chaves a dados biométricos de seu proprietário. Nossa construção também se destaca por ser a primeira assinatura baseada em hash camaleão provada segura neste modelo específico de segurança.

Palavras-chave: Hash Camaleão. Assinaturas Digitais. Provas de Segurança.

RESUMO EXPANDIDO

INTRODUÇÃO

Uma função hash tradicional é um algoritmo que, dado uma mensagem *msg* produz de maneira eficiente uma saída *dgt*, geralmente pequena se comparada com o tamanho da entrada. Além disso, deve ser computacionalmente fácil obter *dgt* a partir de *msg*, mas computacionalmente difícil de obter *msg* a partir de *dgt*. Mais ainda, deve ser também difícil encontrar duas mensagens que resultem na mesma saída.

Ao contrário de funções hash, as funções de hash camaleão são uma primitiva criptográfica que possui associada a ela um par de chaves (*ek*, *tk*). A primeira delas, que chamamos de chave de avaliação e que é pública, torna possível determinar o hash de uma mensagem. A outra chave é um "trapdoor" secreto que torna possível resolver os seguintes problemas que de outro modo seriam difíceis:

- Encontrar duas entradas diferentes que resultam no mesmo hash (computar colisão).
- Dada uma entrada, encontrar uma segunda entrada que resulta no mesmo hash (computar segunda pré-imagem).

Algumas funções de hash camaleão também permitem resolver o problema adicional, que é tão ou mais difícil que os anteriores:

- Dado um valor de hash, encontrar uma entrada da função que produz o valor do resumo (computar pré-imagem).

Neste trabalho fazemos uma revisão de funções de hash camaleão presentes na literatura e apresentamos uma proposta de aplicação alternativa para elas: a criação de assinaturas de pré-imagem. Neste tipo de assinatura, durante a geração das chaves, um usuário pode escolher qualquer elemento arbitrário do conjunto de saídas de uma hash camaleão e associá-la ao par de chaves. O elemento escolhido é revelado durante a verificação da assinatura: o hash de um par formado por uma mensagem e uma assinatura válida resulta no elemento escolhido.

MOTIVAÇÃO

Este trabalho sobre hash camaleão foi motivado pelo estudo de assinaturas digitais e pela ideia de criar uma assinatura digital mais amigável aos usuários.

Em assinaturas digitais típicas, para realizar a sua verificação, um usuário executa um algoritmo que retorna se a assinatura foi considerava válida ou não. O que queremos é que além de termos um algoritmo de verificação, tenhamos também a

possibilidade de extrair determinado símbolo ou informação de uma assinatura válida. Que se uma assinatura for válida, possamos também extrair dela uma imagem da assinatura manuscrita de um signatário para exibi-la na tela, por exemplo.

Como em uma assinatura tradicional tipicamente aplicamos uma função hash tradicional em uma mensagem e então aplicamos um algoritmo de assinatura sobre o resumo, a ideia inicial era buscar implementar um esquema onde o simples cálculo do hash sobre a mensagem e sua assinatura pudesse resultar em uma informação personalizada específica. A qual poderia ser a imagem de uma assinatura manuscrita. Dado que as funções hash tradicionais não permitem que isso seja feito, passamos a buscar funções com maior flexibilidade em suas propriedades. Depois de pesquisar na literatura, concluímos que funções hash camaleão tinham a propriedade que buscamos.

OBJETIVOS

O principal objetivo deste trabalho é o desenvolvimento de uma construção de assinatura digital baseada em hash camaleão. A assinatura desenvolvida deve ter a propriedade de poder codificar uma mensagem ou conteúdo arbitrário que pode ser checado durante a verificação da assinatura caso seja uma assinatura legítima.

Também temos como objetivo garantir que a assinatura seja segura mesmo diante de atacantes quânticos. Com isso espera-se que a assinatura desenvolvida continue segura mesmo com os possíveis avanços futuros na construção de computadores quânticos.

Como objetivos específicos para atingir o objetivo geral acima, podemos listar:

1. Revisar o que existe na literatura sobre hash camaleão.
2. Revisar construções existentes de assinaturas digitais usando funções de hash camaleão.
3. Estudar os requisitos necessários para usar uma função hash camaleão da forma como descrevemos.
4. Propor uma construção específica e eficiente para nossa assinatura.
5. Demonstrar por meio de uma prova ou redução de segurança que nossa assinatura proposta é segura.
6. Implementar um protótipo de nossa assinatura e comparar seu desempenho com o de outras construções.

METODOLOGIA

Como nossa proposta consiste em criar uma assinatura digital a partir da capacidade de computar a pré-imagem de uma hash camaleão com ajuda de seu *trapdoor*, buscamos através de uma revisão sistemática encontrar outros trabalhos que usem esta capacidade presente em algumas funções de hash camaleão (ao invés de usar apenas o cálculo da segunda pré-imagem, que é mais comum). Buscamos assim encontrar casos de uso anteriores para tal propriedade e identificar se essa mesma ideia já foi apresentada antes.

Após investigar o que já existia sobre o assunto na literatura, oferecemos uma proposta nova de construção que pode ser provada segura por meio de uma redução de segurança mostrando que falsificar uma assinatura em nossa construção, mesmo podendo induzir o signatário a assinar mensagens específicas escolhidas adaptativamente, é tão difícil quanto resolver um problema baseado em reticulados. Tal problema é considerado difícil tanto para computadores clássicos como computadores quânticos.

Por fim, após propor uma construção, buscamos oferecer a implementação de um protótipo para demonstrar a viabilidade da construção e poder comparar o seu tempo de execução com o de outras assinaturas digitais.

RESULTADOS E DISCUSSÃO

Nossa revisão sistemática mostrou que embora seja menos comum, existem trabalhos anteriores que exploraram a capacidade de cálculo de pré-imagem de hash camaleão, e inclusive há trabalhos prévios que usam tal capacidade para construir assinaturas digitais. Entretanto, nenhuma das propostas existentes tem a propriedade proposta de permitir codificar informação personalizada arbitrária ligada ao signatário. As construções anteriores também garantiam a segurança apenas em modelos de segurança menos rigorosos.

Uma das dificuldades em usar funções de hash camaleão para tal finalidade é que na maioria das construções, tais funções não permanecem seguras quando colisões são reveladas. Apesar disso, conseguimos propor uma construção capaz de garantir a segurança de nossa construção de assinatura. Na construção proposta, embora não tenhamos provas de que ela permanece segura se colisões arbitrárias forem reveladas, conseguimos garantir que as colisões específicas reveladas durante o processo de assinatura não comprometem a segurança do esquema.

A segurança de nossa construção é baseada na dificuldade de resolver o problema de encontrar vetores pequenos em certos tipos de reticulados, um problema considerado difícil tanto para computadores clássicos como para computadores quânticos.

Apresentamos também uma implementação de um protótipo em C++ que demon-

stra a viabilidade da proposta.

CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho demonstrou a segurança e viabilidade do conceito de assinaturas de pré-imagem por meio de uma prova de segurança e implementação de um protótipo. Com isso, nossa construção poderia ser usada na prática para a assinatura de documentos eletrônicos.

A propriedade mais interessante de nossa assinatura é que ela é parte da pré-imagem de um valor escolhido durante a geração de chaves. Isso abre novas possibilidades, como criar assinaturas onde uma mensagem assinada é verificada comparando seu “hash” depois de concatenada com sua assinatura para verificar se o resultado é um dado valor com interesse especial, como por exemplo, a imagem de uma assinatura manuscrita. Tal uso pode melhorar a experiência do usuário em termos da confiança na assinatura ao permitir que haja não apenas a sua verificação criptográfica, mas também uma verificação visual da mesma.

Outras informações biométricas do signatário também podem ser usadas além da sua assinatura manuscrita. Sua impressão digital, por exemplo. Isso também permite que seja possível ligar as chaves de uma assinatura digital à identidade real de quem irá utilizá-la.

Trabalhos futuros envolvendo a assinatura descrita aqui envolvem pesquisar como aperfeiçoar o seu tempo de execução e investigar se é possível usar outros tipos de construção de hash camaleão em sua composição para que sua segurança possa ser baseado em outros tipos de pressupostos criptográficos. Também pode ser possível generalizar o esquema para que ao assinar uma mensagem o signatário possa escolher uma dentre várias informações personalizadas para associar à sua assinatura, ao invés de ser sempre a mesma.

ABSTRACT

Chameleon hash functions are hash functions with an associated trapdoor that allows for first or second preimage computation. Without the knowledge of this trapdoor, the chameleon hash is collision-resistant like usual hash functions. This thesis presents a review about different constructions of chameleon hash functions and some applications. We review previous uses of chameleon hash functions to build digital signature schemes and propose a novel construction, secure against adaptive chosen message attacks, even against quantum attackers. A security proof in the random oracle model is presented together with results about the scheme implementation. The new proposed signature scheme has an interesting property that a pair composed of a message and a valid signature can encode any arbitrary data chosen during key generation. This encoded data is revealed computing the hash of the message and signature and can be used to create more user-friendly signatures and link the keys to its owner's biometric data. This signature is also notable for being the first signature scheme based on a chameleon hash function proven secure in this specific security model.

Keywords: Chameleon Hash. Digital Signatures. Provable Security.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

The basic building block for secure cryptographic protocols and secure computer systems are cryptographic primitives. These primitives are a collection of algorithms with specific properties considered relevant to achieve desired characteristics such as secrecy, authentication, privacy, integrity and others. Cryptographic primitives should be simple enough to be defined with only a handful of algorithms and security requirements.

Since the beginning of modern cryptography, different primitives such as encryption schemes, digital signature schemes, one-way functions, and collision-resistant hash functions have been proposed and built based on the security of different cryptographic assumptions. The most used are the difficulty of computing discrete logarithms and factoring composite integers. Nevertheless, in addition to these more famous primitives, there are other less-known and studied. Occasionally new primitives are proposed to solve specific tasks.

In 1998, Hugo Krawczyk and Tal Rabin published in (KRAWCZYK; RABIN, 1998) a technical report that suggested a new cryptographic primitive known as “chameleon hash”. The report was published in the Theory of Cryptography Library, an online forum for announcing new works in cryptography. Chameleon hash functions were like regular hash functions with the additional property of having a trapdoor that allows the computation of second-preimages. Informally, if traditional cryptographic hash functions are like a “fingerprint” for digital data, a chameleon hash allows the holder of its trapdoor to produce data with dynamic and adaptable fingerprints that mimic other fingerprints.

Initially, the primitive was proposed to ensure privacy in a specific signature scheme, but new use cases were found and proposed.

Most of this dissertation is about the study of chameleon hash functions: its properties and applications with particular emphasis to their use in building signature schemes.

1.1 MOTIVATION

This work about chameleon hashing was motivated by the study of digital signatures. The author of this dissertation is part of a laboratory that researches in this area and there is interest in alternative properties and constructions for signature schemes. Particularly, in methods for building digital signature schemes which are more user-friendly.

In typical digital signatures, during verification step, an user executes a verification algorithm that returns if the signature is valid or not. What we want is having a signature where besides the verification algorithm, we also have an algorithm able to extract custom information from valid signatures. With such construction, we could

use valid signatures to extract information like an image of the signer's handwritten signature. Such image could be presented in the screen to an user if the signature was accepted by the verification algorithm.

As in a typical signature scheme we first pass the message to a hash function and then apply the signing algorithm to the digest, one idea was if we could create a signature where if we compute the hash of the message and the signature together, we could obtain the chosen custom data (Figure 1). However, this is not possible with usual hash functions. We needed a hash function with some additional flexibility to make this idea workable. After researching the literature, we realized that chameleon hash functions could have the properties we wanted.

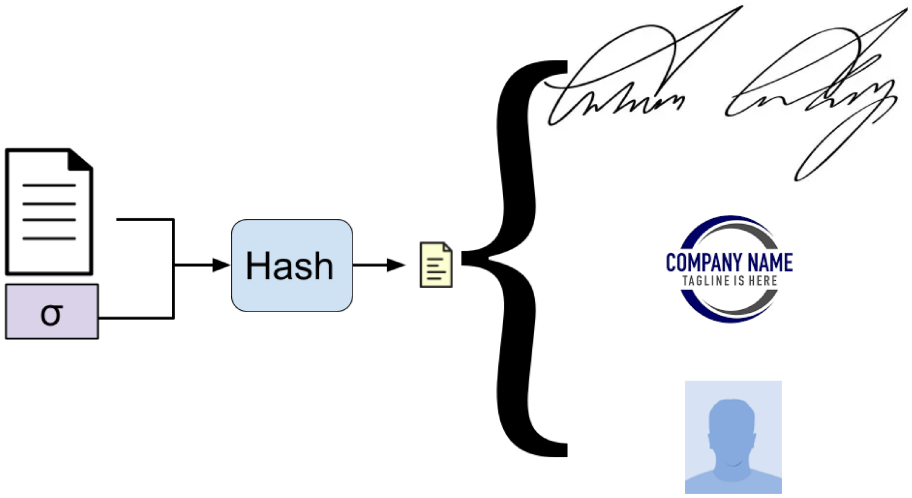

Figure 1 – The motivating idea: having a signature scheme where the signature is valid for a given message only if computing the hash of such message with the given signature yields some custom data, like a image of the signer handwritten signature, a picture or some logo.

1.2 JUSTIFICATION

Traditional digital signature schemes (which were first formally defined in (GOLDWASSER et al., 1988)) allow users to verify the signatures of digitally signed messages by running algorithms that can accept or reject signatures. The user has very little control over the verification process. Simply, the software that implements the verification algorithm shows whether the signature is valid or not. This is very different from checking a handwritten signature on paper. In this case, the user can see if the paper contains what is perceived as the correct signature and can express his opinion on the validity of the signature (more in-depth analysis of this process can be found in(PLAMONDON; SRIHARI, 2000)).

This difference between digital and handwritten signatures makes digital signatures less intuitive for users. To solve this problem, our proposal makes digital signatures

and handwritten signatures coexist and complement each other in the same electronic signature. As in a traditional signature scheme, we have a verifying algorithm. If this algorithm accepts a digital signature, the computer software could extract the image of the handwritten signature to present it to the user. The user is also allowed the opportunity to check the handwritten signature as in a regular non-electronic signature. This could improve the trust of the user in the verification process.

This property also links a pair of keys of a given user to biometric data about this user. In the above example, we mentioned handwritten signatures, but any other arbitrary information also could be encoded into valid signatures, such as fingerprints or pictures. This helps to link the public key of a given user to the identity of a specific person.

1.3 OBJECTIVES

The main objective of this work is to propose a signature scheme that allows the encoding of personalized and arbitrary data in our signatures using chameleon hash functions.

Additionally, due to the imminent availability of quantum computers, which supposedly could break some of the difficult computational problems used in cryptographic primitives, we want that our new proposed signature be quantum attack-proof. Such schemes are known in the literature as post-quantum algorithms.

1.3.1 Specific Objectives

To achieve the general objective presented above, the following specific objectives are listed:

1. **Literature review about chameleon hash functions.** Study different constructions and properties about chameleon hash functions. Check if a chameleon hash suitable for our objectives is already described in the literature.
2. **Literature review about signatures built using chameleon hash functions.** Check if a signature with the desired property already was proposed in the literature. Furthermore, if not, learn what other ideas about using chameleon hash functions in signature construction were already explored.
3. **Find the requisites necessary for using a chameleon hash in our construction.**
4. **Propose a new digital signature scheme.** This new scheme should be based on chameleon hash functions, should be secure against post-quantum attackers and should have the special property of encoding a custom information chosen during key generation in valid signatures.

5. **Write a security proof for the proposed scheme.** This involves presenting a security model for our proposal. Our proof should be compatible with post-quantum security.

6. **Implement the proposed scheme.** With an implemented prototype, we can compare the performance of our proposal with other constructions. Moreover, demonstrate the viability of the construction.

1.4 METHODOLOGY

Initially, we studied the first works on chameleon hash functions. The first work published was in 1998. Since that date, its definition and use cases have been improved until approximately 2006, from when there were no changes in the basic concept behind this cryptographic construction. As a result of this study, we describe precisely the primitive and its possible uses.

This initial study showed that most papers that propose applications for chameleon hash functions describe them as hash functions where one can find second-preimages using the trapdoor. However, we needed a more powerful variant where one can compute preimages, not only second-preimages.

Motivated by this, we have done a systematic review looking for papers in the last ten years (2010-2020) that mention chameleon hash functions and preimages. Our review is presented in Annex A. With this search, we evaluated in each work whether the chameleon hash concept whose trapdoor allows the calculation of the first preimage had been considered and, if so, for which use cases.

Using the previous survey results, we also checked the works referenced in them looking for additional research on signature construction using chameleon hash functions. This procedure yielded some more results that were studied.

After studying the current techniques on signature construction and chameleon hash functions secure against quantum attackers, we propose to build our signature scheme using a new variant of lattice-based chameleon hash functions. We built a prototype of this new signature scheme in a C++ implementation based on an existing code that implemented the lattice-based GPV signature scheme. Furthermore, using this implementation, we compared the performance of our initial construction with other known signatures, both post-quantum signatures and classical ones.

1.5 SCIENTIFIC CONTRIBUTIONS

In this work, we propose a generalisation of the concept of chameleon hash, which we call preimage chameleon hash. We use this to build a new signature scheme and show how to adapt the post-quantum chameleon hash from (CASH et al., 2010) to make our signature secure. We prove our signature scheme to be strongly unforgeable

under a chosen message attack (SUF-CMA). The new signature scheme is presented in chapter 6 which resulted in the following paper:

- ASTRIZI, Thiago; CUSTÓDIO, Ricardo; MOURA, Lucia. Post-quantum signature with preimage chameleon hashing. In: SBC. XX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais, 2020

1.6 NOTATION

In this work, we use lower case Latin letters, like a, to represent integer values. Upper case Latin letters represent sets. Real numbers, possibly non-integers, are represented by Greek letters like β. Bold lower-case letters, like $\mathbf{a}$, are vectors and bold upper-case letters, like $\mathbf{A}$ are matrices. Polynomials are represented by lower-case letters followed by parenthesis like in $p()$.

In all the above cases, all elements can have subscripts. For example a_1 and b_2 are integer numbers. The element R_q is a set and $\mathbb{A}_1$ is a matrix.

Additional notation and some exceptions for the above conventions, when necessary, are introduced next to the texts where they are used.

1.7 ORGANIZATION

The remainder of this dissertation is organized as follows.

In Chapter 2, we describe the mathematical background for security proofs and provide a list of cryptographic primitives and cryptographic assumptions that are referenced in the rest of the document.

In Chapter 3, describes chameleon hashing, the definition, security requirements, some existing constructions and applications.

In Chapter 4, we introduce a more powerful version of chameleon hash functions, whereby using a trapdoor, it is possible to find first preimages, not only second preimages. We list existing constructions, properties and applications.

In Chapter 5, we present signature constructions using chameleon hash functions.

In Chapter 6, we present our novel signature construction and the results about its implementation.

In Chapter 7, we finish with a conclusion and further works.

2 PRELIMINARIES

This chapter is organized as follows:

- Section 1 introduces the concept of security proofs, what are cryptographic assumptions, and security reductions.
- Section 2 defines a list of cryptographic assumptions derived from hard mathematical problems used in this work.
- Section 3 defines a list of cryptographic primitives that we use in this work.
- Section 4 is about non-standard models useful in some security proofs. We describe the random oracle model, a model used in several security proofs in this dissertation.

2.1 PROOFS OF SECURITY

Having unconditional security proof for a cryptographic construction is hard, and few primitives have such proof. For example, most cryptographic constructions must assume that $P \neq NP$ or it would be impossible to guarantee its security. However, despite the enormous interest in this fundamental question, finding proof that $P \neq NP$ (or that $P = NP$) remains an open problem.

Nevertheless, even if one managed to prove that $P \neq NP$, this would be only the beginning step necessary to prove the security of most cryptographic constructions. If $P \neq NP$, then we can find problems whose solutions can be verified in polynomial time but cannot be solved in the worst case in polynomial time. However we need hard problems in cryptography that are hard not only in the worst case, but on average case. Therefore, we also need proof that there are problems that are hard on average.

Finally, after proving that there exist problems that can be easily verified, but are hard to be solved on average, we would also need to prove the existence of one-way functions: functions that are easy to compute but hard to invert. With this proof and knowing a function that is one-way without doubts, we could have definitive proofs for some cryptographic constructions and primitives. Nevertheless not for all of them: while we know necessary primitives that can be built directly from one-way functions (signature schemes, pseudo-random functions), there is no known construction of some other primitives (hash functions) based only on one-way functions.

To illustrate all these different steps, in the seminal paper (IMPAGLIAZZO, 1995), the author Russel Impagliazzo described five different worlds, each one holding a different number of the above assumptions:

- **Algorithmica:** In this world, $P = NP$. Its inhabitants can solve in polynomial time any problem which could be verified in polynomial time. Most known cryptography

is useless in this world. However, we still could use the few cryptographic constructions with unconditional security: one-time pads and some universal hash functions, for example. However, keys cannot be shared in any untrusted channel.

- **Heuristica:** In this world $P \neq NP$. There are problems that cannot be solved in polynomial time in the worst case. Though, it is very tough to find these worst cases. All problems in *NP* are easy in the average case. This creates a scenario in which the hardness of a problem is proportional to the time and resources that we spent looking for the problem. As we use cryptography as protection against adversaries with more resources than us, most cryptographic techniques also are useless in this world.

- **Pessiland:** Here $P \neq NP$ and it is easy to find hard instances of problems. However, one-way functions do not exist. We cannot find hard problems whose solutions we know simply working backward from the solution to the instance of the problem. Finding hard problems is easy, but nobody will know the solution to these problems. This is considered the worst world by Impagliazzo, as most cryptography is still impractical in this world, and at the same time, the computers in Pessiland are not much more powerful than ours at solving most of the problems.

- **Minicrypt:** Secure one-way functions exist. Therefore, we know that there are also other secure primitives like pseudo-random generators (PRG), pseudo--random functions (PRG), stream ciphers and some signature schemes. Yet, using only one-way functions, we still do not know how to build secure collision-resistant hash functions and asymmetric cryptography. So we cannot share secrets in a untrusted public channel.

- **Cryptomania:** Both one-way functions and one-way functions with trapdoor exist. This implies the existence of public key cryptography and the possibility of exchanging secret messages over open and insecure channels. It is widely believed that this is the world where we live.

The five worlds do not encompass all the possibilities. Between Minicrypt and Cryptomania, we can have intermediary worlds depending on the specific properties of the secure one-way functions known and depending of what other cryptographic primitives besides one-way functions are possible. And there are suggestions about adding another world called **Obfustopia** to represent a scenario where we also have indistinguishable obfuscation: we can obfuscate functional computer programs such that they cannot be distinguished from any other program with the same size and functionality (GARG et al., 2016). What these five worlds description illustrate is how far we are from having unconditional proofs for most of cryptographic primitives. We do

not even prove that we do not live in Algorithmica, Heuristica or Pessiland, where most of the existing cryptography is entirely insecure.

Because of this difficulty, modern cryptography assumes conditional security and conditional security proofs, also known as security reductions. We assume that some specific problem (our cryptographic assumptions) is hard on average and cannot be solved in polynomial time. Assuming that this is true, then we prove that our cryptographic construction is secure. Usually, this involves using proof by contradiction: given an adversary that breaks the security of our construction, we create an adversary that solves the problem that we assume hard.

We have two kinds of cryptographic assumptions. The first kind is based on hard mathematical problems. We usually trust that they are hard problems not because we have a proof, but for historical reasons. Some of these problems were studied for a long time, and despite the amount of time and energy spent trying to solve them, they remain without a solution. The second kind of assumption is about the existence of some cryptographic primitive. For example, collision-resistant hash functions or one-way functions. The following section lists assumptions of the first kind and Section 2.3 will list assumptions of the second type.

When we say that a problem is hard on average, this means that for some specific challenge modeled as an attacking game between a challenger and some adversary $\mathcal{A}$, the probability of $\mathcal{A}$ succeed and win that game is negligible. Both the challenger and the adversary are a sequence of algorithms that share an internal state. The specifications for both the challenger and the adversary are given by a specific attack game defined for each security property.

When we say that the probability of $\mathcal{A}$ succeeds in an attack game is negligible, we say that for any polynomial $p(\lambda)$, the following equation holds:

$$\lim_{\lambda \to \infty} p(\lambda) Pr[\mathcal{A} \text{ succeeds}] = 0$$

The probability $Pr[\mathcal{A}$ succeeds$]$ is in function of the security parameter λ. We say that any function that behaves as defined above is a **negligible function**. We also use two other related definitions:

Definition 1 *We say a function $f : \mathbb{N} \to \mathbb{R}$ is **superpolynomial** if $1/f(\lambda)$ is a negligible function.*

Definition 2 *We say a function $f : \mathbb{N} \to \mathbb{R}$ is **polynomially bounded** or **limited polinomially** if for some polynomial $p(\lambda)$ and some constant c, for all $\lambda > 0$ it is true that $|f(\lambda)| \leq \lambda^c$.*

From properties of limits, we have that:

- If $f(n)$ and $g(n)$ are negligible, then $(f + g)(n)$ and $(f \cdot g)(n)$ are also negligible.

- If $f(n)$ and $g(n)$ are polynomially bounded, then $(f+g)(n)$ and $(f \cdot g)(n)$ are also polynomially bounded.
- If $f(n)$ and $g(n)$ are superpolynomial, then $(f+g)(n)$ and $(f \cdot g)(n)$ are also superpolynomial.
- If $f(n)$ is negligible and $g(n)$ is polynomially bounded, then $(f+g)(n)$ is polynomially bounded and $(f \cdot g)(n)$ is negligible.
- If $f(n)$ is negligible and $g(n)$ is superpolynomial, then $(f+g)(n)$ is superpolynomial.
- If $f(n)$ is polynomially bounded and $g(n)$ is superpolynomial, then $(f+g)(n)$ and $(f \cdot g)(n)$ are superpolynomial.
- If for all values n greater than some constant, if $f(n) \leq g(n)$ and g is negligible, then f is negligible.

2.2 MATHEMATICAL CRYPTOGRAPHIC ASSUMPTIONS

Here we list the cryptographic assumptions that we will use in this work. The notation of how we describe them is based on the book (BONEH; SHOUP, 2020).

2.2.1 Factoring Assumption

Let GENPRIME be an efficient algorithm that takes as input a positive integer $n > 1$ and outputs a randomly chosen prime number with n bits. As prime numbers are relatively common, this algorithm can generate random odd numbers and return the first result that pass in a primality test.

Now we will describe the factoring assumption with the help of the above algorithm and the following attack game:

Attack Game 1 ***(Factoring Assumption).*** *We have an adversary $\mathcal{A}$ and a challenger. Both are initialized with the security parameter λ. First, the adversary runs the algorithm* GENPRIME(λ) *until having two different primes: p and q.*

Next, the challenger computes $z = pq$ and send z to the adversary $\mathcal{A}$. And the adversary sends as response a tuple (p', q'). We say that the adversary $\mathcal{A}$ wins the game if $p' > 1$, $q' > 1$ and if $p'q' = z$.

The probability of a given adversary $\mathcal{A}$ winning this game is denoted by FACTadv[$\mathcal{A}$].

The factoring assumption says that for all efficient adversaries $\mathcal{A}$, the value of *FACTadv*[$\mathcal{A}$] is negligible (as a function of λ).

It is widely believed that factoring large semiprimes is hard if we only consider classical algorithms. In the RSA Factoring Challenge, a challenge created by RSA

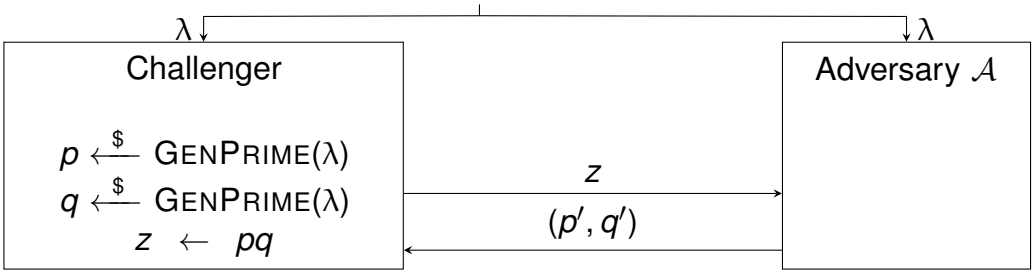

Figure 2 – Attack Game: Factoring Assumption

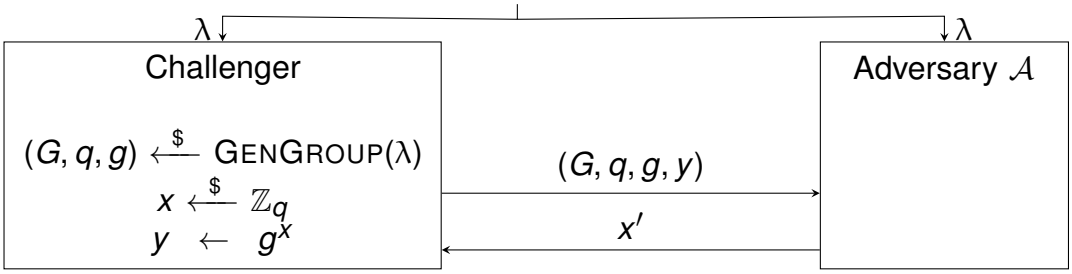

Figure 3 – Attack Game: Discrete Logarithm

Laboratories that offered prizes for successful factorization of semiprimes, the biggest factored number had 829 bits and was the multiplication of two random primes with 415 bits. Therefore, the biggest λ for which the problem was solved is $\lambda = 415$ ((ZIMMERMAN, 2020)). All known classical algorithms solve the factoring problem only in superpolynomial time.

The factoring assumption is known to be false when we consider a post-quantum scenario. A quantum algorithm for factoring semiprimes in polynomial time is proposed in (SHOR, 1994).

2.2.2 Discrete Logarithm Assumption

Let GENGROUP be an algorithm invoked as $(G, q, g) \xleftarrow{\$}$ GENGROUP(λ) where G is a description of a cyclic multiplicative group with order q (where q is a λ-bit number) and $g \in G$ is a generator. It means that $G = \{g^0, g^1, \ldots g^{q-1}\}$. We also require that the multiplication in the group can be performed in polynomial-time. Consider the following attack game:

Attack Game 2 ***(Discrete Logarithm).*** *The challenger and the adversary are initialized with the security parameter. The challenger runs* GENGROUP(λ) *to obtain the tuple* (G, q, g)*. Then it chooses an integer x uniformly at random between 0 and $q-1$ and computes $y = g^x$. It sends $(G, 1, g, y)$ to the adversary.*

The adversary returns an element x'. It wins the game if $g^{x'} = y$. We represent as DLadv$[\mathcal{A}, G']$ the probability of a given adversary $\mathcal{A}$ wins this game, assuming that GENGROUP *produces groups from the set G'.*

The discrete logarithm assumption says that there exist a family of groups G' and a corresponding algorithm GENGROUP such that for all efficient adversaries $\mathcal{A}$, the value of $DLadv[\mathcal{A}, G']$ is negligible.

Usually, when using this assumption, we assume that the order q is a prime number. For composite numbers, if we can factor them, we could solve the discrete logarithm reducing one instance of this problem to two smaller instances using the Pohlig-Hellman algorithm, making the problem easier than expected (see (BACH, 1984)). Another advantage of prime orders is that any element besides the identity could be chosen as the generator g.

Some candidate cyclic groups can be used in this problem:

- The multiplicative group $\mathbb{Z}_q^*$ or one of its subgroups if q is not prime.
- The group composed of points in discrete elliptic curves.

In (BACH, 1984) it was proven that given an adversary $\mathcal{A}$ able to compute discrete logarithm in the cyclic group $\mathbb{Z}_n^*$, then it is possible to build an adversary $\mathcal{B}$ able to factor n. Therefore, for all efficient adversaries $\mathcal{A}$ that can win the discrete logarithm attack game for groups in the form $\mathcal{Z}_n^*$, there exists an efficient adversary $\mathcal{B}$ able to win the factoring attack game such that:

$$DLadv[\mathcal{A}, \mathbb{Z}_n^*] \leq FACTadv[\mathcal{B}]$$

When we have such property given two cryptographic assumptions, we say that the assumption at the right side of the above inequation is stronger. In our case, this means that the factoring assumption is stronger than the discrete logarithm assumption. It is preferred to base the security of cryptographic constructions using the weakest possible assumption, as we assume that less unproven assumptions are true in our conditional proofs.

The biggest discrete logarithm in $\mathbb{Z}_n^*$ solved in a competition had a order with 795 bits (BOUDOT et al., 2020) ($\lambda = 795$).

All known classical algorithms solve the discrete logarithm problem in superpolynomial time. But like the factoring problem, we know quantum algorithms to solve this problem in polynomial time and therefore the discrete logarithm assumption is false in post-quantum scenarios (SHOR, 1994).

2.2.3 Short Integer Solution (SIS) Assumption

This assumption relies on the difficulty of finding vectors of integers with a sufficiently short euclidean norm such that when multiplied by a known integer matrix chosen uniformly at random from $\mathbb{Z}_q^{n \times m}$ result in a vector filled with zeros.

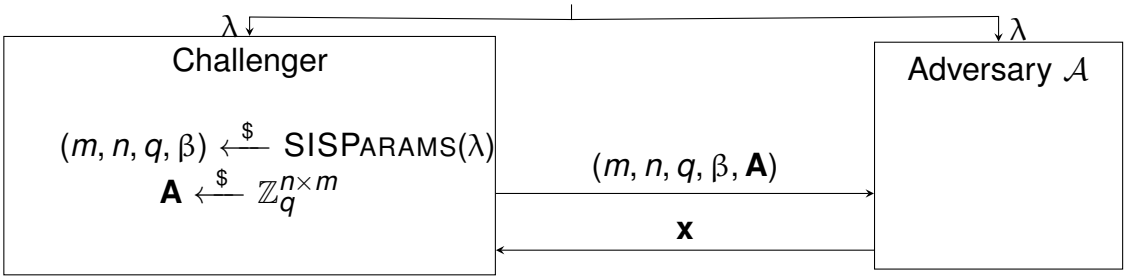

Figure 4 – Attack Game: Short Integer Solution

Assume that we have an algorithm SISPARAMS(λ) that return a tuple (m, n, q, β). The first two integers will determine the matrix size. The third is the modulus q that limits the possible integers in the matrix. Moreover, the last value β is what we consider a sufficiently small euclidean norm.

With this we define the following attack game:

Attack Game 3 ***(Short Integer Solution).*** *Let a challenger and an adversary $\mathcal{A}$ be initialized by the security parameter λ. The challenger derives secure values (m, n, q, β) using algorithm* SISPARAMS *and chooses uniformly at random a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$. It sends to the adversary the tuple $(m, n, q, \beta, \mathbf{A})$.*

Adversary $\mathcal{A}$ sends as response a nonzero vector $\mathbf{x} \in \mathbb{Z}^m$. We say that adversary $\mathcal{A}$ wins the game if the euclidean norm of the vector (denoted by $||\mathbf{x}||$) is smaller that β and $A\mathbf{x} = 0 \pmod q$.

We denote by SISadv$[\mathcal{A}]$ as the probability of a given adversary $\mathcal{A}$ win this game.

According to the SIS assumption, there exists an algorithm SISPARAMS such that for all adversaries $\mathcal{A}$, the value of *SISadv*$[\mathcal{A}]$ is negligible.

To avoid trivial solutions and at same time ensure that a sufficient number of solutions exist, the parameters are chosen such that $\beta < q$ (or a trivial solution $(q, 0, 0, \ldots, 0)$ can be returned), n is polynomially bounded by the security parameter, $\beta \geq \sqrt{n \log q}$ and $m \geq n \log q$.

This assumption was proposed for the first time in the article (AJTAI, 1996). In that paper, the author proves the relationship between this assumption and some problems believed to be hard involving lattices. If one could find an adversary $\mathcal{A}$ that wins the above attack game, it could also use $\mathcal{A}$ to build an algorithm that solves some lattice problems believed to be hard in the worst case. For example, finding a sufficiently short vector different than zero that is the linear combination of other vectors. That was the first time that a cryptographic assumption was shown to relate to the difficulty of a known mathematical problem in the worst case, not in the average case as is usual.

Another advantage of this assumption is that it is believed to be true even when considering quantum algorithms. There are no known classical or quantum algorithms that can solve the SIS problem in polynomial time. More about the security of this

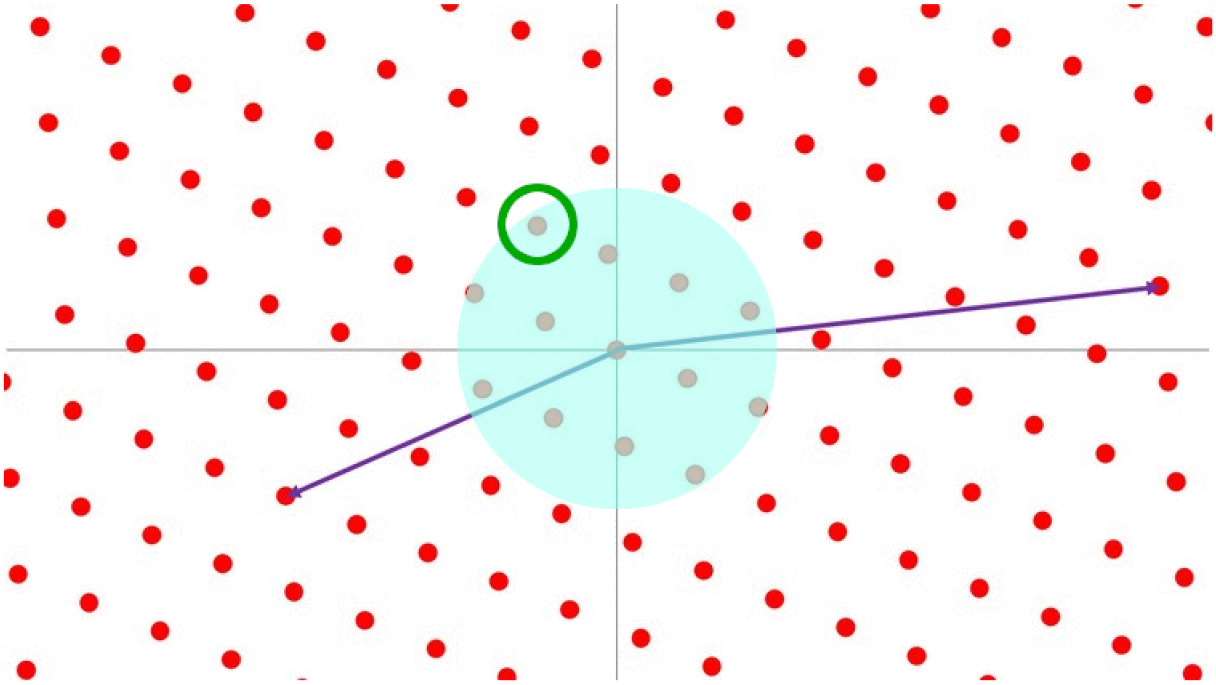

Figure 5 – Solving the Short Integer Solution is finding a sufficient short vector in a lattice (the dots in the blue area in the above image). A lattice is defined by a linear combination with integer coefficients of vectors (shown in purple in this image). Solving this problem is believed to be hard when the number of dimensions is big.

assumption and its resistance against quantum attacks can be seen in (PEIKERT, 2016).

While usually the SIS assumption is described using the euclidean norm, as we did above, there are other possible norms that can be used and are believed to be safe. For example, instead of using the euclidean norm in a vector like in $||\mathbf{x}||$, we can define the following additional norm for $\mathbf{x} = (x_1, \ldots, x_n)$:

$$||\mathbf{x}||_\infty = max(\{|x_1|, \ldots, |x_n|\})$$

. More about the use of alternative norms can be find in (PEIKERT, 2008).

And for the two alternative norms above, we denote respectively by $SISadv_1[\mathcal{A}]$ and $SISadv_\infty[\mathcal{A}]$ the probability that an adversary $\mathcal{A}$ wins the SIS attack game using these alternative norms instead of the euclidean norm.

2.2.4 Ring Short Integer Solution (Ring-SIS) Assumption

This assumption can be seen as a variation of the SIS assumption. Instead of using integer matrices and vectors, we use a polynomial ring R_q and vectors of elements in this ring.

Let $R = \mathbb{Z}[x]/(f(x))$ be a polynomial ring. The elements of this ring are polynomials modulo $f(x)$. And let R_q be polynomial ring modulo $f(x)$ where all the coefficients

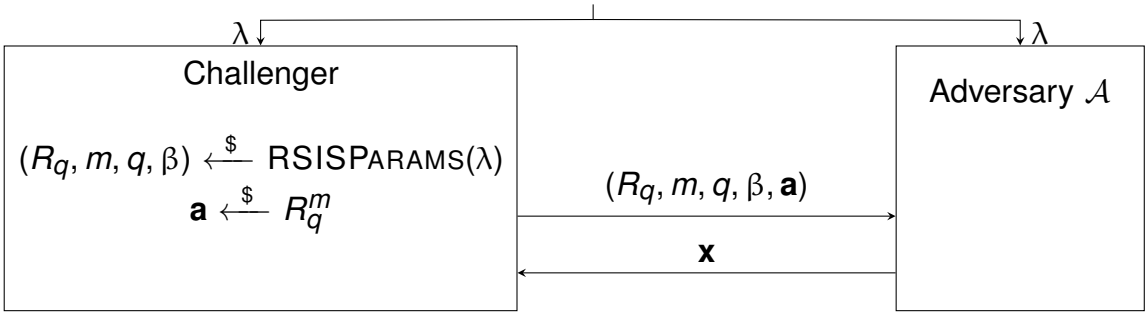

Figure 6 – Attack Game: Ring-SIS Assumption

are integers modulo q.

Let $p(x)$ be a polynomial with coefficients $a_1, \ldots, a_n$. We define its euclidean norm as $||p(x)|| = \sqrt{\sum_{i=1}^{n}(a_i)^2}$. And the euclidean norm of a vector of polynomials $\mathbf{v} = (p_1(x), \ldots, p_m(x))$ is given by $||\mathbf{v}|| = \sqrt{\sum_{i=1}^{n}(||p_i(x)||)^2}$.

As in the SIS assumption, assume that we have an algorithm RSISPARAMS(λ) that return a tuple (R_q, m, q, β). The first element is a description of a polynomial ring. The integer m will be the size of the vectors in our problem. The integer q will be our modulus. Furthermore, β is the maximum euclidean norm of our vectors.

With this, we can define our new attack game:

Attack Game 4 ***(Ring-SIS Assumption).***

Let a challenger and an adversary $\mathcal{A}$ be initialized by λ. The challenger gets (R_q, m, q, β) running RSISPARAMS(λ)*. Next, it produces a random and uniform vector $\boldsymbol{a} \in R_q^m$.*

The challenger sends to the adversary the tuple $(R_q, m, q, \beta, \boldsymbol{a})$. The adversary replies with another nonzero vector $\boldsymbol{x} \in R^m$.

We say that the adversary wins the game if $||\boldsymbol{x}|| < \beta$ and the scalar multiplication of $\boldsymbol{a}$ and $\boldsymbol{x}$ is equal to 0. We denote by RSISadv$[\mathcal{A}]$ the probability of a given adversary $\mathcal{A}$ wins this game.

According to the Ring-SIS assumption, there exists an algorithm RSISPARAMS for which for any adversary $\mathcal{A}$, the value of *RSISadv*$[\mathcal{A}]$ is negligible.

This assumption is very similar to the SIS assumption. One of its attractiveness is that the vector of polynomials **a** can be smaller than the matrix **A** from the SIS assumption. Therefore, cryptographic constructions based on the Ring-SIS assumption usually has smaller keys and are faster.

Succeeding at the Ring-SIS attack game is shown in (LANGLOIS; STEHLÉ, 2015) to be equivalent to succeeding in a variation of the SIS attack game where the matrix **A** is chosen uniformly at random from a subset of matrices with a circulant structure, where each column is a rotation of the first column and where rotated elements are multiplied by -1.

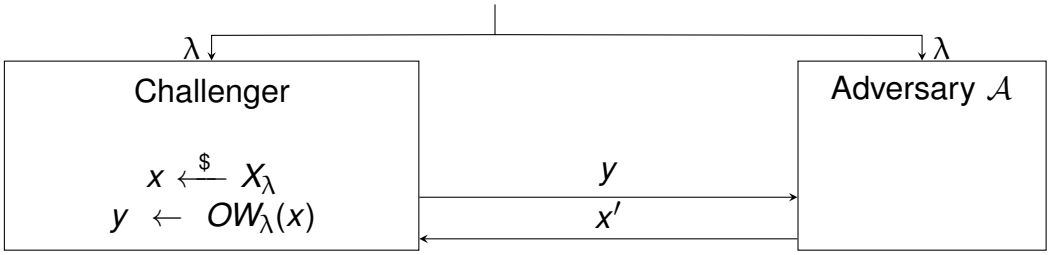

Figure 7 – Attack Game: One-Way Function

As for the SIS assumption, there are no known quantum or classical algorithms that solve the challenge of its attack game in polynomial time ((PEIKERT, 2016)).

2.3 CRYPTOGRAPHIC PRIMITIVES

2.3.1 One-Way Function

A one-way function $OW : X \to Y$ is a function such that given some element y in its codomain, it is hard to find an element $x \in X$ such that $OW(x) = y$. Defining using an attack game, we have:

Attack Game 5 ***(One-Way Functions).*** *For a family of one-way functions, we have an adversary and a challenger initialized by a security parameter λ. Each security parameter λ properly identifies one one-way function $OW_\lambda : X_\lambda \to Y_\lambda$ in the family.*

The challenger chooses a random and uniform element $x \in X_\lambda$ and computes $y = OW_\lambda(x)$. It sends y to the adversary that outputs some value x'.

We say that an adversary $\mathcal{A}$ wins the game if $OW_\lambda(x') = y$. Note that not necessarily $x = x'$.

We denote by $OWadv[\mathcal{A}, OW]$ the probability of a given adversary $\mathcal{A}$ winning this attack game for a given family of one-way functions OW.

We say that some one-way function family OW is secure if $OWadv[\mathcal{A}, OW]$ is negligible for all efficient adversaries $\mathcal{A}$.

Except in more formal definitions, we usually keep implicit the security parameter λ when using one-way functions. If we write $OW(x)$, we assume that we choose a one-way function from some suitable family with an appropriate security parameter.

The existence of one-way functions is also a cryptographic assumption. We do not know if they exist. If we surely know that they exist, we also would know that $P \neq NP$. Even so, one-way functions are very fundamental primitives. It is easy to build them using any of the cryptographic assumptions from the previous section. If the discrete logarithm assumption is true, then $OW(x) = g^x$ is a one-way function given a suitable group and a generator g. If the SIS assumption is true, $OW(\mathbf{x}) = \mathbf{Ax} \mod q$ is a one-way function given a random and uniform matrix $\mathbf{A}$.

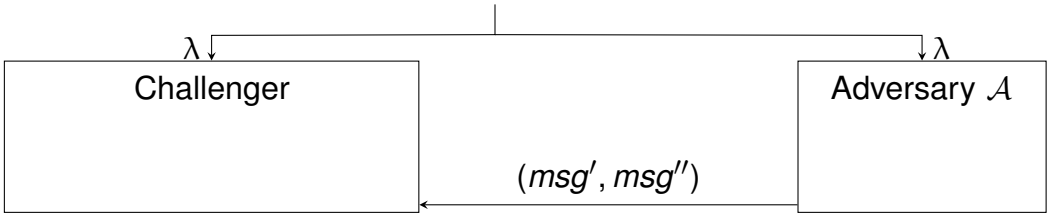

Figure 8 – Attack Game: Collision-resistance.

If for some one-way function $OW : X \to Y$ we have that $X = Y$ and that OW is bijective, we say that OW is an **one-way permutation**.

For all one-way functions, we can define an adversary $\mathcal{A}$ that randomly chooses a polynomial number of elements in the domain and checks for each x_i obtained this way if $OW(x_i)$ is equal to the challenge y received as input. The probability of this adversary succeeding in the attack game is at least $p(\lambda)/|Y_\lambda|$ for some polynomial $p(\lambda)$. Therefore, for all one-way function families OW, we can always build an efficient adversary $\mathcal{A}$ such that:

$$\frac{p(\lambda)}{|Y_\lambda|} \leq OWadv[\mathcal{A}, OW]$$

The probability is increased if the codomain is smaller than its image. Therefore, in a secure one-way function, the cardinality of its codomain (and therefore also the cardinality of $|X|$) must be superpolynomial.

2.3.2 Hash Function

A (non-chameleon) hash function is a function $HASH : M \to D$ where $|D| < |M|$ such that the function is **collision-resistant**. Informally this means that it is difficult to find distinct $msg_1, msg_2 \in M$ such that $HASH(msg_1) = HASH(msg_2)$. For a more formal definition, we use the following attack game:

Attack Game 6 ***(Collision Resistance).*** *For a family of hash functions, we have an adversary and a challenger, both initialized with the security parameter λ. Each security parameter λ properly identifies each hash function $HASH_\lambda : M_\lambda \to D_\lambda$ in the family.*

The adversary outputs a pair (msg', msg'') and we say that it wins the game if $msg' \neq msg''$ and if $HASH(msg') = HASH(msg'')$.

We define $\mathcal{A}$'s advantage concerning to the family of hash functions HASH as the probability of $\mathcal{A}$ winning the game. We denote this advantage as $CRadv[\mathcal{A}, HASH]$.

We say that a hash function family is collision-resistant if for all efficient adversaries $\mathcal{A}$, we have that $CRadv[\mathcal{A}, HASH]$ is a negligible value.

As for one-way functions, except in more formal definitions, we keep the security parameter λ implicit and use $HASH(msg)$ directly assuming that we choose some $HASH$ family from some family using an appropriate security parameter.

For any hash function *HASH*, we always can build an efficient adversary that chooses a polynomial $p(\lambda)$ number of messages from M_λ, for each message msg_i with $0 \leq i < p(\lambda)$, it computes $dgt_i \leftarrow HASH_\lambda(msg_i)$, and finally checks if some of the generated output is a collision. This is called a **birthday attack**. We can find a lower bound for the probability that this attack succeeds, assuming that the output of *HASH* is randomly distributed (if not, the probability increases).

Using the bounds from (WIENER, 2005), we have that for all hash function families, the upper bound for the probability of not finding a collision for the above attack is $e^{-\frac{p(\lambda)(p(\lambda)-1)}{2|D_\lambda|}}$. Using this and the fact that for all real $-1/2 \leq x \leq 0$ we have that $(1 + x) \geq e^{2x}$, we can conclude that for all hash function families *HASH* we can build an efficient $\mathcal{A}$ performing a birthday attack such that:

$$\frac{p(\lambda)(p(\lambda)-1)}{4|D_\lambda|} \leq CRadv[\mathcal{A}, HASH]$$

The above inequality is true only if $|D_\lambda| > p(\lambda)(p(\lambda)-1)$ (as we used in the bound the fact that the exponent in e was between 0 and $-1/2$). However, we can see that the left side is negligible only if $|D_\lambda|$ is superpolynomial, which for sufficiently big values of λ, ensures that the exponent is in this range. Therefore, the above lower bound is always true for collision-resistant hash functions, and a hash function is collision-resistant only if the cardinality of $|D|$ is superpolynomial.

2.3.2.1 Weaker Properties: Second-Preimage

Sometimes we are interested in functions with properties weaker than collision resistance. One of these properties is the second preimage. While the collision-resistance says that it is hard to find any two messages (msg', msg'') such that $HASH(msg') = HASH(msg'')$, the second-preimage says that it is hard to find some message msg' that collides with a given random message. More formally, we define this property with the following attack game:

Attack Game 7 ***(Second preimage resistance).*** *For a family of hash functions, we have an adversary and a challenger. Both are initialized with the security parameter λ. Each security parameter λ properly identifies each hash function $HASH_\lambda : M_\lambda \rightarrow D_\lambda$ in the family.*

The adversary selects uniformly at random a message: $msg \xleftarrow{\$} M_\lambda$ and sends msg to the adversary.

The adversary outputs msg' and we say that it wins the game if $msg' \neq msg$ and if $HASH(msg') = HASH(msg)$.

We define $\mathcal{A}$'s advantage with respect to the family of hash functions HASH as the probability of $\mathcal{A}$ winning the game. We denote this advantage as $SPREadv[\mathcal{A}, HASH]$.

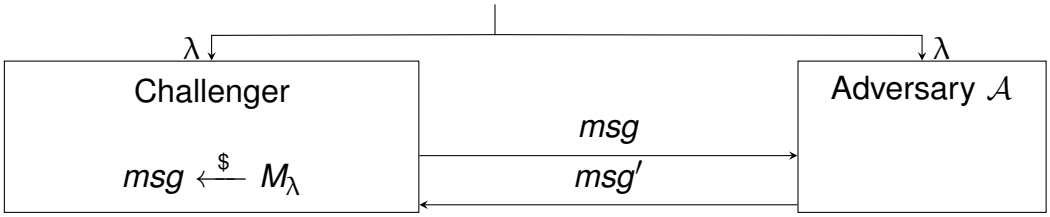

Figure 9 – Attack Game: Second preimage resistance.

A hash function family *HASH* is **second preimage resistant** if $SPREadv[\mathcal{A}, HASH]$ is negligible for all adversaries $\mathcal{A}$.

We can show that this property is weaker than the collision-resistance with the help of the following theorem.

Theorem 1 *All collision-resistant families of functions are also second-preimage resistant.*

Proof: Assuming that we have an adversary $\mathcal{A}$ that interact with the adversary trying to find a second preimage. Let's use it to build a new adversary $\mathcal{B}$ that finds collisions in the same functions:

- Adversary $\mathcal{B}(\lambda)$:
  1. $msg \xleftarrow{\$} M_\lambda$
  2. $msg' \xleftarrow{\$} \mathcal{A}(\lambda, msg)$
  3. **return** (msg, msg')

The above adversary finds a collision if the adversary $\mathcal{A}$ succeeds at finding a second preimage. If $\mathcal{A}$ fails, then $\mathcal{B}$ fails too. Therefore, for all efficient adversaries $\mathcal{A}$, we can build efficient $\mathcal{B}$ such that:

$$SPREadv[\mathcal{A}, HASH] \leq CRadv[\mathcal{B}, HASH]$$

As by hypothesis, *HASH* is collision-resistant, the right side of this inequation is negligible. Therefore, the left side must also be negligible, which proves the second-preimage resistance of *HASH*. ■

It is important to notice that the converse is not necessarily true. For example, the MD5 and SHA1 constructions of hash functions are known not to be collision resistant. However, there is no known method to find second preimages in them.

2.3.2.2 Weaker Properties: First Preimage Resistance

Another related property is the **preimage resistance**. This property already was defined in Subsection 2.3.1, it is the same security property used for one-way functions.

Usually, a second-preimage resistant hash function is also a one-way function, but not always. If the function compresses the input very little, breaking the preimage resistance does not break the second-preimage resistance or the collision-resistance. If there are no existing collisions for most of inputs, it is possible that computing preimages do not help much in finding collisions.

To better explain this concept, we define the **compression factor** of a given function $HASH : M \rightarrow D$ as $\frac{|M|}{|D|}$. And we prove the following theorem:

Theorem 2 *If a family of functions is second-preimage resistant and has a superpolynomial compression factor, it is also a one-way family of functions.*

Proof: As before, we show how we can build an adversary $\mathcal{B}$ that tries to find second-preimages if we have an adversary $\mathcal{A}$ that can find preimages:

- Adversary $\mathcal{B}(\lambda, msg)$:
  1. $dgt \leftarrow HASH_\lambda(msg)$
  2. $msg' \xleftarrow{\$} \mathcal{A}(\lambda, dgt)$
  3. **return** msg'

  Let s be the compression factor.

The above adversary wins the game finding a second-preimage if adversary $\mathcal{A}$ succeeds and if $msg' \neq msg$. The first probability is $OWadv[\mathcal{A}, HASH]$. About the second probability, it is not independent of $\mathcal{A}$. In the worst case for our adversary $\mathcal{B}$, $\mathcal{A}$ will succeed to compute preimages only if the digest *dgt* has a single message mapped to it by our *HASH* function. In this case, $\mathcal{B}$ never will succeed, even if $\mathcal{A}$ sometimes succeeds.

Fortunately, in a *HASH* function with compression factor s, we will have $|D|$ digests and $s|D|$ different messages. The number of messages without second preimages is less than $|D|$. Therefore, the probability that a random message do not have a second preimage is less than $\frac{|D|}{s|D|} = \frac{1}{s}$.

Therefore, the probability of $\mathcal{A}$ producing a correct result for a digest obtaining a message with existing preimages will be in the worst-case better than $OWadv[\mathcal{A}, HASH]$–$\frac{1}{s}$.

However, even if $\mathcal{A}$ succeeds in winning its game for some messages that can collide with other messages, in the worst possible scenario, it will do so only for messages with a minimal number of collisions. In the worst case, this number will be 2. In this case, the probability that $msg = msg'$ will be 1/2. This means that $\mathcal{B}$ will succeed with probability that in the worst case will be $\frac{1}{2}(OWadv[\mathcal{A}, HASH] - \frac{1}{s})$. Rewriting this, we have that for all adversaries $\mathcal{A}$, we can build an adversary $\mathcal{B}$ such that:

$$OWadv[\mathcal{A}, HASH] \leq 2 \cdot SPREadv[\mathcal{B}, HASH] + \frac{1}{s}$$

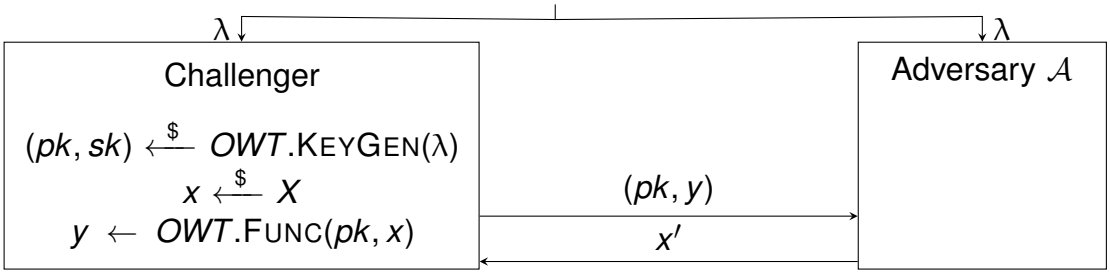

Figure 10 – Attack Game: One-way trapdoor function.

As we assume that *HASH* is resistant to second-preimage, $2 \cdot SPREadv[\mathcal{B}, HASH]$ is negligible. Also, by our hypothesis, s is superpolynomial. Therefore, $1/s$ is negligible and the right side of the above inequation is negligible. If $OWadv[\mathcal{A}, HASH]$ is always less than some negligible value, independent of the adversary $\mathcal{A}$, this means by definition that *HASH* is a one-way function. ■

2.3.3 One-Way Trapdoor Function

A one-way trapdoor function is a scheme *OWT* defined over sets (X, Y) formed by algorithms ($OWT.\text{KEYGEN}$, $OWT.\text{FUNC}$, $OW.\text{INV}$) such that:

- $OWT.\text{KEYGEN}$ is a probabilistic algorithm invoked as $(pk, sk) \xleftarrow{\$} OWT.\text{KEYGEN}(\lambda)$ for some security parameter λ. We call pk the public key and sk the secret key.
- $OWT.\text{FUNC}$ is a deterministic algorithm invoked as $y \leftarrow OWT.\text{FUNC}(pk, x)$ with $x \in X$. The output is an element $y \in Y$.
- $OWT.Inv$ is a deterministic algorithm invoked as $x \leftarrow OWT.\text{INV}(sk, y)$ with $y \in Y$. The output is a $x \in X$ such that $OWT.\text{FUNC}(pk, x) = y$.

The security of this primitive is defined using the following attack game:

Attack Game 8 ***(One-way trapdoor function security).*** *For a given one-way trapdoor function scheme OWT defined over sets (X, Y) we have an adversary $\mathcal{A}$ and a challenger, both initialized by λ.*

The challenger runs $OWT.\text{KEYGEN}$ to get a pair of keys, chooses a random and uniform $x \in X$, computes $y \leftarrow OWT.\text{FUNC}(pk, x)$, and sends (pk, y) t the adversary $\mathcal{A}$.

The adversary outputs $x' \in X$ to the challenger and we say that it wins the game if $OWT.\text{FUNC}(pk, x') = y$. We denote by $OWadv[\mathcal{A}, OWT]$ the probability of a given adversary wins this game.

2.3.4 Pseudo-Random Functions

A pseudo-random function is a cryptographic primitive defined over sets (K, X, Y) and composed of a function $PRF : K \times X \to Y$. The set K is the key space. If we see

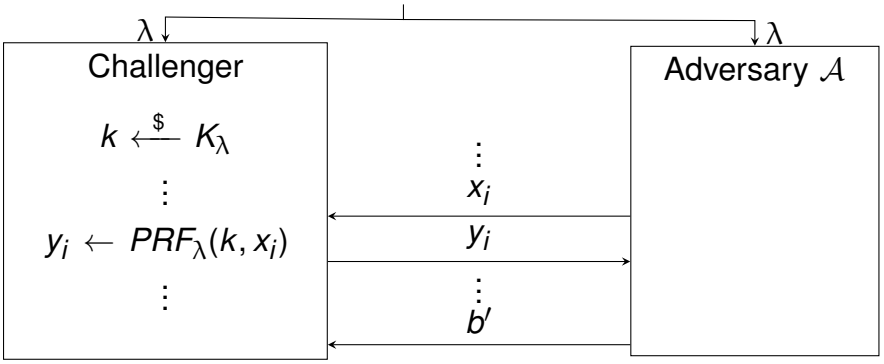

Figure 11 – Attack Game: Pseudo-random function, game 0.

this function applied to a polynomially bounded number of elements $x \in X$ using a fixed key $k \in K$ chosen uniformly at random, we cannot distinguish between this function $PRF(k, \cdot)$ and a completely random function that maps elements of X to Y.

Let K, X, and Y be finite sets. Let $Func[X, Y]$ denote the set of all functions that map elements of X to Y. We define the security of a given PRF with the help of two different attack games.

Attack Game 9 ***Pseudo-Random Function Security)(Game 0).*** *For a given pseudo-random function family PRF, let an adversary $\mathcal{A}$ and a challenger be initialized by a security parameter λ. Each λ identifies a function PRF_λ in the family defined over sets $(K_\lambda, X_\lambda, Y_\lambda)$.*

The challenger chooses a random and uniform key k from K_λ. Next, the challenger can make a polynomially bounded number of queries q_p to the challenger. In each query it sends some element $x_i \in X_\lambda$. The challenger computes $y_i = PRF(k, x_i)$ and sends it to the adversary.

After all the queries, the adversary outputs a single bit $b' \in \{0, 1\}$.

The above attack game represents the adversary interacting with the PRF. In the next one, it interacts with a completely random function.

Attack Game 10 ***Pseudo-Random Function Security)(Game 1).*** *For a given pseudo-random function family PRF, let an adversary $\mathcal{A}$ and a challenger be initialized by a security parameter λ. Each λ identifies a function PRF_λ in the family defined over sets $(K_\lambda, X_\lambda, Y_\lambda)$.*

The challenger chooses a random and uniform function f from $Func[X, Y]$. Next the challenger can make a polinomially bounded number of queries q_p to the challenger. In each query it sends some element $x_i \in X_\lambda$. The challenger computes $y_i = f(x_i)$ and send y_i to the adversary.

After all the queries, the adversary outputs a single bit $b' \in \{0, 1\}$.

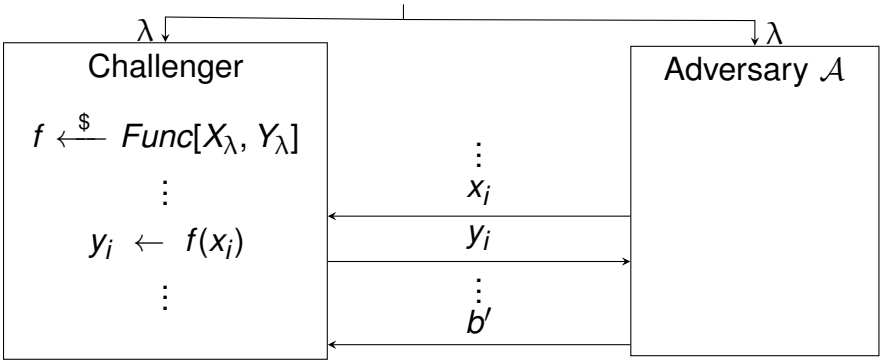

Figure 12 – Pseudo-random function, game 1

In a secure *PRF* no adversary should be able to distinguish between game 0 and game 1. For some adversary $\mathcal{A}$ and a pseudo-random function, we denote by $Pr[W0]$ the probability of $\mathcal{A}$ outputting bit 1 in the attack game 0, and we denote by $Pr[W1]$ the probability of $\mathcal{A}$ outputting the bit 1 in the attack game 1. We denote the advantage of adversary $\mathcal{A}$ as:

$$PRFadv[\mathcal{A}, PRF] = |Pr[W0] - Pr[W1]|$$

We say that a *PRF* is secure or indistinguishable from a random function if for all adversaries $\mathcal{A}$, the value of $PRFadv[\mathcal{A}, PRF]$ is negligible.

We can also define a weaker security requirement for pseudo-random functions. If we modify the previous attack games to forbid queries sent by the adversary, instead of making the challenger choose a polynomially bounded number of random and uniform elements from X and sending the adversary a tuple with pairs $(x_i, PRF(k, x_i))$ (in game 0) and $(x_i, f(x_i))$ (in game 1). If a *PRF* is secure in this modified attack game, we call it a weak pseudo-random function. And we denote its security in this model as $WPRFadv[\mathcal{A}, PRF]$.

2.3.5 Signature Scheme

A signature scheme *SIG* is a tuple of efficient algorithms (*SIG*.KeyGen, *SIG*.Sign, *SIG*.Verify) defined over two sets (M, S) such that:

- *SIGN*.KeyGen is a probabilistic algorithm that takes the security parameter λ as input and outputs a pair (pk, sk) where pk is the **public key** and sk is the **secret key**.
- *SIGN*.Sign is a probabilistic algorithm invoked as $sig \xleftarrow{\$} SIGN.\text{Sign}(sk, msg)$ where sk is a secret key, $sig \in S$ and $msg \in M$.
- *SIGN*.Verify is a deterministic algorithm invoked as $SIGN.\text{Verify}(pk, msg, sig)$. Here $msg \in M$ and $sig \in S$. It outputs either `accept` or `reject`.

- Given a pair of keys (pk, sk) returned by $SIGN.\text{KeyGen}$, for all $msg \in M$ we have:

$$Pr[SIGN.\text{Verify}(pk, msg, SIGN.\text{Sign}(sk, msg)) = \texttt{accept}] = 1$$

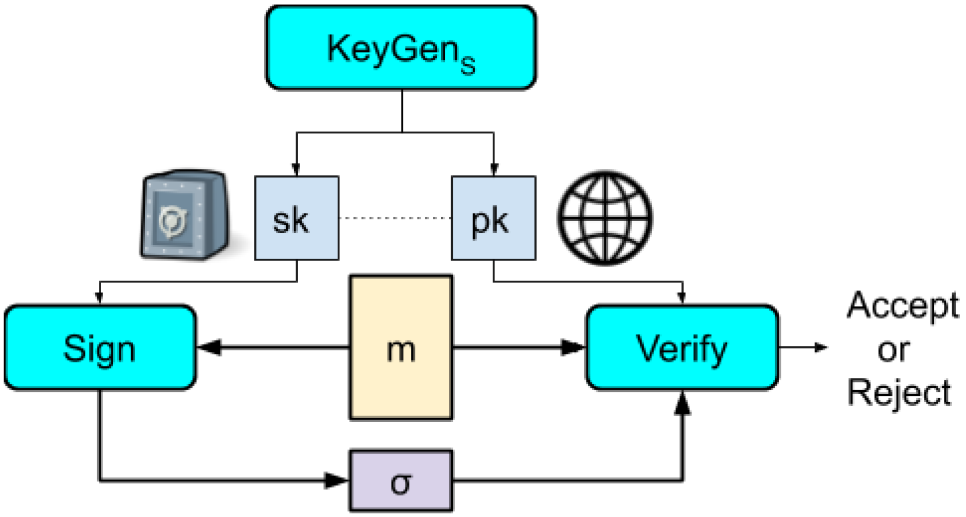

Figure 13 – Diagram representing the algorithms in a signature scheme.

The security of a signature scheme usually uses the following attack game:

Attack Game 11 ***(Signature security against adaptive chosen message attack).*** *For a given signature scheme SIG, consider an adversary $\mathcal{A}$ and a challenger, both initialized by λ.*

The challenger computes a pair of keys (pk, sk) running $SIG.\text{KeyGen}(\lambda)$ and sends pk to the challenger.

Next, the adversary can send a polynomially bounded number of queries for the challenger. Each query is a message $msg_i \in M$. For each one, the challenger runs $sig_i \leftarrow SIG.\text{Sign}(sk, msg_1)$ and send sig_i to the adversary. After all the queries, the adversary output a pair (msg', sig').

We say that an adversary $\mathcal{A}$ wins the game if $SIG.\text{Verify}(pk, msg', sig') = \texttt{accept}$ and if for all queried messages msg_i, we have $msg' \neq msg_i$. We denote by $CMAadv[\mathcal{A}, SIG]$ the probability of an adversary $\mathcal{A}$ wins this game for the signature scheme SIG.

If for the signature scheme *SIG* we have that for all adversaries $\mathcal{A}$ the value of $CMAadv[\mathcal{A}, SIG]$ is negligible, then we say that the signature is unforgeable under a chosen message attack.

The number of signing queries q_s allowed usually is assumed to be a very large number despite being limited polynomially. However, some signatures require smaller values. For the signatures known as **one-time signatures** we have $q_s = 1$. We denote

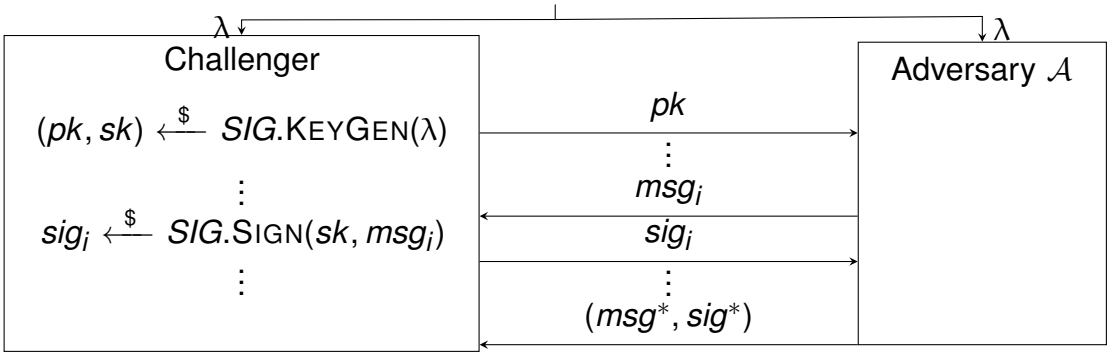

Figure 14 – Attack Game: Signature unforgeability against adaptive chosen message attacks

the probability that an adversary $\mathcal{A}$ wins the attack game performing a single signing query by $CMAadv_1[\mathcal{A}, SIG]$.

2.3.5.1 Weaker Security Notion: Generic Chosen Message Attack

The previous security definition can be relaxed. In the previous attack game, the adversary could query any message to the challenger, and choose the next query based on the results of previous queries. For some signature schemes, we can prove its security only if we disallow these adaptive queries.

In this case, we can use the following weaker attack game:

Attack Game 12 ***(Signature security against generic chosen message attack).*** *For a given signature scheme SIG defined over* (M, S)*, let an adversary* $\mathcal{A}$ *and a challenger be initialized by the security parameter* λ*.*

The challenger computes a pair of keys (pk, sk) *using* $SIG.\text{KEYGEN}(\lambda)$ *and sends pk to the adversary. The adversary chooses a polynomially bounded number of* q_s *messages* $(msg_1, \ldots, msg_{q_s})$ *and sends this tuple of messages to the challenger. The challenger computes a signature for each of these messages with* $SIG.\text{SIGN}(sk, \cdot)$ *and send a tuple of signatures* $(sig_1, \ldots, sig_{q_s})$ *to the adversary.*

After this single exchange of communication, the adversary outputs a forgery (msg', sig')*. We say that the adversary wins this game if* $SIG.\text{VERIFY}(pk, msg', sig') =$ `accept` *and if the pair* (msg_i, sig_i) *is different than all pairs* (msg_i, sig_i) *with* $i \in [1, q_s]$*. We denote by* $GCMAadv[\mathcal{A}, SIG]$ *the probability of some adversary* $\mathcal{A}$ *wins this game against a signature scheme SIG.*

We can define some variations in the above attack game as we did for the adaptive chosen message attack. For some signature constructions, we can prove the security only in a variant attack game where the adversary makes non-adaptive queries before knowing the public key. The challenger sends the public key *pk* with the signature tuple $(sig_1, \ldots, sig_{q_s})$. We say that in this case, the unforgeability and security of our

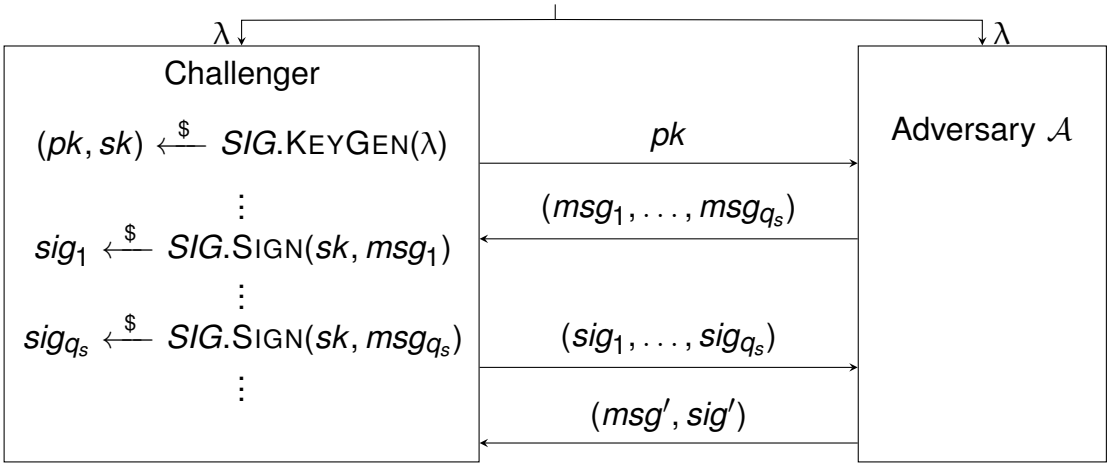

Figure 15 – Attack Game: Signature unforgeability against generic chosen message attacks

signature are weak. Furthermore, we denote by $GCMAadv_{Weak}[\mathcal{A}, SIG]$ the probability of a given adversary succeed in this attack game. If the signature is also a one-way signature, we denote it by $GCMAadv_{1\text{-}Weak}[\mathcal{A}, SIG]$.

It is easy to see that if we have an adversary $\mathcal{A}$ that can create forgeries in this generic chosen message attack, it also can be modified to create forgeries in the adaptive chosen message attack. It can produce all its signing queries initially, but instead of sending all of them in a single tuple, it sends each query individually as required by the adaptive chosen message attack game. Therefore, given a single signature scheme *SIG*, for all adversaries $\mathcal{A}$, we can create adversaries $\mathcal{B}$ such that:

$$GCMAadv[\mathcal{A}, SIG] \leq CMAadv[\mathcal{B}, SIG]$$

Being secure against adaptive chosen message attack is a stronger property. We can also see that an adversary that can create forgeries in an attack game that defines weak security for the signature can also be adapted to succeed in the stronger security attack game. It need only ignore the public key *pk* at the beginning of the attack game. Therefore, given a single signature scheme *SIG*, for all adversaries $\mathcal{A}$, we can create adversaries $\mathcal{B}$ such that:

$$GCMAadv_{Weak}[\mathcal{A}, SIG] \leq GCMAadv[\mathcal{B}, SIG]$$

And also, if we can create a forgery for a one-time signature before knowing its public key *pk*, we could also create forgeries in the scenario where we have this information:

$$GCMAadv_{1\text{-}Weak}[\mathcal{A}, SIG] \leq CMAadv_1[\mathcal{B}, SIG]$$

2.3.5.2 Weaker Security Notion: Random Message Attack

Finally, the weakest security definition for signatures that we will consider is the security against random message attacks. In this security model, the adversary

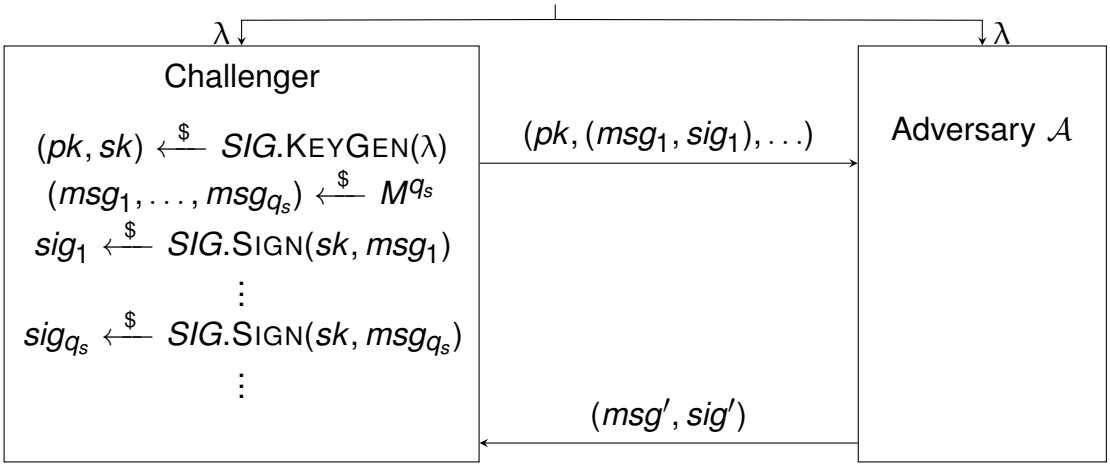

Figure 16 – Attack Game: Signature unforgeability against random message attacks

cannot choose the messages for signing queries. The challenger computes signatures for messages chosen uniformly at random and sends for him. We define this security model with the next attack game:

Attack Game 13 ***(Signature security against random message attack).*** *For a given signature scheme SIG defined over* (M, S)*, let an adversary* $\mathcal{A}$ *and a challenger be initialized by the security parameter* λ*.*

The challenger computes a pair of keys (pk, sk) *using* $SIG.\text{KEYGEN}(\lambda)$ *and also chooses a polynomially bounded number of* q_s *messages chosen uniformly at random from the message space M.It signs each of these messages using* $SIG.\text{SIGN}(sk, \cdot)$ *and send to the adversary the key pk and a tuple with all pairs of randomly chosen messages and its signatures.*

After this, the adversary outputs a forgery (msg', sig')*. We say that the adversary wins this game if* $SIG.\text{VERIFY}(pk, msg', sig') = \texttt{accept}$ *and if the pair* (msg_i, sig_i) *is different than all pairs* (msg_i, sig_i) *with* $i \in [1, q_s]$*. We denote by* $RMAadv[\mathcal{A}, SIG]$ *the probability of some adversary* $\mathcal{A}$ *wins this game against a signature scheme SIG.*

We say that a signature *SIG* is unforgeable or secure against a random message attack if for all adversaries $\mathcal{A}$, the value of $RMAadv[\mathcal{A}, SIG]$ is negligible.

If we have an adversary $\mathcal{A}$ that creates forgeries against *SIG* in a random message attack, we can adapt it to also create forgeries in a generic chosen message attack. It only needs to choose a total of q_s messages chosen uniformly at random and send these messages to the adversary. Therefore, for all adversaries $\mathcal{A}$ in a random message attack, we can create an adversary $\mathcal{B}$ such that:

$$RMAadv[\mathcal{A}, SIG] \leq GCMAadv[\mathcal{B}, SIG]$$

2.4 NON-STANDARD MODELS IN SECURITY PROOFS

Some cryptographic primitives are very difficult to be proven secure using standard techniques. This challenge is common for signature schemes or public-key encryption schemes. In such cases, non-standard models can be used, where part of the construction is idealized. Proofs using these models bring security guarantees weaker than proofs in the standard model. Nevertheless such proofs still are useful, as they attest the security of the construction for a relevant category of attackers. For example, attackers that treat the result of a hash function as random.

2.4.1 Random Oracle Model

This model was introduced in (BELLARE; ROGAWAY, 1993). It treats hash functions as completely random functions. For a function $HASH : M \rightarrow D$, we model it as some function chosen uniformly at random from $Func[M, D]$.

Notice that this is an idealization. No real-life hash function will ever behave like this. Informally, we could program a completely random hash function creating an array with the number of elements in M, and in each position, we could put an element from D chosen uniformly at random. The hash of any message $msg_i \in M$ would be the element in position i of this array. The space complexity of this construction is superpolynomial. However, we cannot improve this for a random function. As each element of this array is random, our array cannot be compressed.

Creating a computer program shorter than this huge array representing our hash function by itself compresses the array and, therefore, cannot be done for most of the possible random arrays. In other words, the Kolmogorov complexity of a completely random hash function would be too big.

Despite not being possible to implement a real random oracle function $\mathcal{O}$, we can simulate them using lazy evaluation techniques. We start with an empty dictionary, and each time we want to check the value of $\mathcal{A}(msg_i)$, we check if the key msg_i is stored in the dictionary. If so, we use the value stored with it. If m_i is not a key in the dictionary, we generate a random and uniform $dgt_i \in D$, store it in using m_i as the key and assume that $dg_i = \mathcal{O}(msg_i)$. This simulation is how we can trick adversaries in the random oracle model that they are dealing with a random oracle, when in fact, they are interacting with an efficient algorithm.

All attack games in the previous section can be adapted to the random oracle model. The difference is that at the beginning of the game, the challenger chooses uniformly at random a random oracle $\mathcal{O} \in Func[M, D]$ for each hash function that maps M to D. More than one random oracle can be chosen if some cryptographic construction uses more than one hash function. Every time the challenger needs to compute some function $HASH_i$, it uses the corresponding random oracle $\mathcal{O}_i$ instead. As

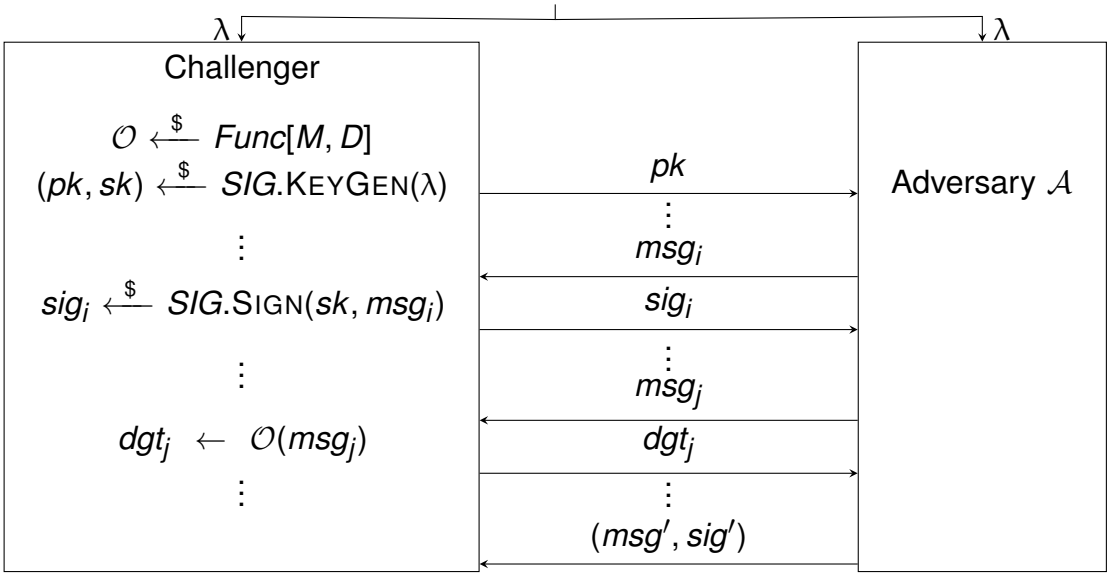

Figure 17 – Attack Game: Signature unforgeability against adaptive chosen message attacks in a random oracle model

it is expected that an adversary should compute hash functions, we let the adversary send a polynomially bounded number of q_O random oracle queries to the challenger. In each query it sends some $msg_i \in M$ and gets as response $\mathcal{O}(msg_i)$.

As an example, we present in Figure 17 how the attack game for signature security against adaptive chosen message attacks would be modified in a random oracle model. The query for message msg_i represents the same signing query that we already had. The query for message msg_j is a random oracle query and represents the access that the adversary has to the hash function. Notice that both kinds of queries can be intercalated in any order.

We denote the probability of a given adversary $\mathcal{A}$ succeed in that attacking game against a signature *SIG* by $CMAadv^{RO}[\mathcal{A}, SIG]$. For all other attack games, if we are using the random oracle model, we also append the suffix ^{RO} to distinguish probabilities in this model from probabilities in the standard model.

To see how one could use the random oracle model to prove the security of a cryptographic construction, we describe next a signature scheme known as Full-Domain Hash Signature and prove its security in a random oracle model.

2.4.1.1 Proving Full-Domain Hash Signature in the Random Oracle Model

Consider the signature scheme known as Full-Domain Hash, denoted by SIG_{FDH}. We will use a collision-resistant function function $HASH : M \to D$ and a one-way trapdoor function *OWT* that is also a trapdoor permutation function and is defined over the set (D, D). We define the signature scheme SIG_{FDH} defined over sets (M, D) with the following algorithms:

- SIG_{FDH}.KEYGEN(λ):
  1. $(pk, sk) \xleftarrow{\$} OWT$.KEYGEN($\lambda$)
  2. **return** (pk, sk)
- SIG_{FDH}.SIGN(sk, msg):
  1. $dgt \leftarrow HASH(msg)$
  2. $sig \leftarrow OWT$.INV(sk, dgt)
  3. **return** sig
- SIG_{FDH}.VERIFY(pk, msg, sig):
  1. $dgt' \leftarrow OWT$.FUNC(pk, sig)
  2. **if** $dgt' = HASH(msg)$:
  3. **return** `accept`
  4. **else**:
  5. **return** `reject`

This signature works because with the help of a trapdoor *sk* we can find inverses for the function *OWT*.FUNC($pk, \cdot$). Moreover, as this one-way function is a permutation in this case, for each possible digest returned by *HASH* for some message, there is a single valid signature.

To prove the security of this signature, we need the random oracle model.

Theorem 3 *If the one-way trapdoor function OWT is secure, then the signature SIG_{FDH} built using it is secure against adaptive chosen message attacks in the random oracle model.*

Proof: To prove this theorem, we will assume that we have an adversary $\mathcal{A}$ that can create forgeries for SIG_{FDH} after interacting with its challenger in the random oracle model, as modeled by Figure 17. Using this adversary $\mathcal{A}$, we will show how to create an adversary $\mathcal{B}$ that break the security of *OWT* as described by the one-way trapdoor function attack game (seen in Figure 10).

First, we build an adversary $\mathcal{B}$ that can trick adversary $\mathcal{A}$ int thinking that it is interacting with a legitimate adversary. Even without knowing the trapdoor of the *OWT* function and without implementing a real random oracle, adversary $\mathcal{B}$ will interact with $\mathcal{A}$ in a way indistinguishable from a legitimate challenger from a one-way trapdoor function attack game. For now, adversary $\mathcal{B}$ is not producing a correct output, only tricking $\mathcal{A}$.

We build adversary $\mathcal{B}$ in the following way:

- **Initialization**: First adversary $\mathcal{B}$ is initialized with the security parameter λ and then it gets from its challenger the public key *pk* of the one-way trapdoor function and some value *y*. This *pk* by itself is a valid public key for the signature SIG_{FDH}. Then, adversary $\mathcal{B}$ initializes adversary $\mathcal{A}$ with the security parameter λ and with the signature public key *pk*. To help simulate a random oracle efficiently, adversary $\mathcal{B}$ initializes an empty dictionary that will store pairs of messages and signatures.

- **Random Oracle Query**: Given a random oracle query for the message msg_j, adversary $\mathcal{B}$ first checks in the dictionary if msg_j already have a known signature. If checking the dictionary, it sees that the signature for message msg_j is sig_j, it can deduce the value of $\mathcal{O}(msg_j)$ computing $OWT.\text{FUNC}(pk, sig_j)$ and sending this as a response. If it do not know the signature for msg_j, first it chooses uniformly at random some signature $sig_j \in D$, store this in the dictionary as the signature for msg_j and compute $OWT.\text{FUNC}(pk, sig_j)$ to produce the response for the random oracle query.

  Notice that the response for a random oracle produced in this way is indeed random and uniform because we are obtaining it by choosing a random and uniform value (the signature) and applying this over a permutation function (the one-way trapdoor function). Therefore, the adversary $\mathcal{A}$ cannot notice that it is not interacting with a correct challenger, the responses produced by $\mathcal{B}$ are indistinguishable from a legitimate challenger.

- **Signing Query:** When $\mathcal{A}$ sends a signing query for message msg_i, adversary $\mathcal{B}$ first checks in the dictionary if the message msg_i is already stored with a corresponding signature. If so, it sends as a response the stored signature. If not, it produces a signature sig_i choosing some value from *D* uniformly at random.

  Notice that in the random oracle model, our signatures are computed using $OWT.\text{INV}(sk, \mathcal{O}(msg_i))$. The output of $\mathcal{O}(msg_i)$ is random for unknown messages msg_i and $OWT.\text{INV}(sk, \cdot)$ is also a permutation in our case. Therefore, choosing a signature randomly $\mathcal{B}$ produces a response for the query that is indistinguishable from a response produced by a legitimate challenger.

- **Finalization:** In the end, adversary $\mathcal{A}$ produces a forgery (msg', sig'). Adversary $\mathcal{B}$ checks if the signature was successful. If msg' is in the dictionary, $\mathcal{A}$ succeeds if sig' is the signature stored there and msg' never was queried in a signing query. If it is not in the dictionary, we generate a random and uniform signature, and adversary $\mathcal{A}$ wins if this signature is equal to sig'.

Note that our objective with $\mathcal{B}$ is not only tricking $\mathcal{A}$, but finding the inverse of *y* that is received during the initialization. By definition, this value *y* is chosen uniformly at random (as if $OWT.\text{FUNC}(pk, \cdot)$ is a permutation, then *y* is this permutation applied

to a value chosen uniformly at random). So for one of the random oracle queries instead of producing the response described above, we could send y. If we send y as a response to a random oracle query for message msg_j and in the end, we have a forgery (msg', sig') such that $msg' = msg_j$, this means that sig' is the inverse of y and $\mathcal{B}$ can output this to win its attack game.

However, this assumes that we correctly guess wich random oracle query has a message used in the forgery. If we guess wrong, not only adversary $\mathcal{B}$ will not be able to find the preimage of y given a correct forgery, but also there is the risk that $\mathcal{A}$ sends the same message in a signing query and $\mathcal{B}$ will not be able to give a correct response. If the forgery (msg', sig') has a message msg' that was queried before in a random oracle query, we have a probability of $1/q_s$ of guessing correctly which random oracle query will be used in the forgery.

However, there is also the possibility that $\mathcal{A}$ do not perform a random oracle query previously with msg'. We can try to guess this scenario too. If we guess correctly in this case, we assume that $\mathcal{O}(msg') = y$, and we keep the property that if we guessed correctly, then the signature sig' from the forgery is a preimage of y.

Considering that $\mathcal{A}$ can produce a forgery using a message sent in any one of the q_O random oracle queries, or can produce a forgery using a message never queried before, this gives adversary $\mathcal{B}$ a total of $q_O + 1$ possible choices when guessing what will happen. It can make the guess generating a random and uniform value between 1 and $q_O + 1$. If it gets a random oracle query that it assumes by it guess that will be used in a forgery, it can sent y as response to this query. Or if it guesses that the final forgery will use a message msg' never queried before, it assumes that $\mathcal{O}(msg') = y$. The guess not necessarily will be correct. In fact, it will have only a probability of $1/(q_O + 1)$ of being correct.

With this change, we have an adversary $\mathcal{B}$ that succeed if we have the following two events:

- Adversary $\mathcal{A}$ succeed at creating a forgery. This happens with a probability given by $CMAadv^{RO}[\mathcal{A}, SIG_{FDH}]$.
- We guessed correctly which random oracle query was used in the forgery or if the forged message never was sent in a random oracle query. This happens with probability $\frac{1}{q_O+1}$.

The two probabilities above are independent. Therefore, the lower bound for the probability of success for $\mathcal{B}$ is given by:

$$OWTadv[\mathcal{B}, OWT] \geq \frac{1}{q_O + 1} CMAadv^{RO}[\mathcal{A}, SIG_{FDH}]$$

Rewriting the above inequation, we have that for all adversaries $\mathcal{A}$ that forge signatures in SIG_{FDH} in the random oracle model, we can build an adversary $\mathcal{B}$ that

find a preimage for the one-way trapdoor function *OWT* such that:

$$CMAadv^{RO}[\mathcal{A}, SIG_{FDH}] \leq (q_O + 1)OWTadv[\mathcal{B}, OWT]$$

By our assumption, the one-way trapdoor function *OWT* is secure. Therefore $OWTadv[\mathcal{B}, OWT]$ is always negligible for all adversaries. Furthermore, the number of random oracle queries is polynomially bounded (even if it is modeled by a polynomial with a very high degree). Multiplying a negligible value by a polynomially bounded value, we have a negligible value. Therefore, the upper bound for $CMAadv^{RO}[\mathcal{A}, SIG_{FDH}]$ is negligible, and therefore, no adversary can forge signatures in this model, except with negligible probability. ■

2.4.1.2 Random Oracle in a Post-Quantum Model

In the post-quantum model, we assume that the adversaries in the attack game can use efficient quantum algorithms, not only classical algorithms. In this model, cryptographic assumptions like discrete logarithm or factoring are false, as there are quantum algorithms that can solve them. However, other than using assumptions that hold against quantum attackers, this usually does not change much in the standard model.

When an attacker and a challenger exchange messages, we assume that they communicate using a classical channel, not quantum. So it is not necessary to deal with typical quantum scenarios where we get in our query a superposition of a super-polynomial number of messages.

However, in a random oracle model, the oracle queries represent access to a hash function, and in a realistic scenario, a quantum adversary should be able to perform queries with a superposition of exponentially many messages.

This problem is described in (BONEH et al., 2011) where the author listed the following scenarios that invalidate security proof in the random oracle model against quantum adversaries:

1. Adaptive Programmability: If we simulate a random oracle needing to check information from the past to choose each new query response adaptively correctly, this technique cannot be used in proof for a post-quantum scenario. A quantum adversary can perform each query sending a superposition of many messages, and we would need to reply in such queries a response with information about many queries at once. This also forbids us to encode a relevant value in our queries, as we did in the previous proof sending y as one response.

2. Preimage Awareness: If we simulate a random oracle to know beforehand possible values used in the final output of our quantum adversary and we need to use this information in our proof, then the proof is not valid for quantum adversaries. If a quantum adversary can send a super-polynomial number of messages in

superposition for each query. We cannot guess with non-negligible probability which message will be used in the adversary output.

3. Efficient Simulation: We cannot simulate a random oracle using lazy evaluation. If we could get a superpolynomial number of messages in superposition in a single random oracle query, we need to know beforehand the response for each query, we cannot generate the responses on the fly, as we did in the previous query.

4. Rewinding: We cannot execute twice the same quantum adversary and expect that it will produce the same outputs, even if we feed it with the same input in both executions. For quantum adversaries, this would violate a property known as the no-cloning theorem. Quantum algorithms can be inherently non-deterministic. This restriction also applies to proofs in the standard model, not only for proofs in the random oracle model.

These restrictions are not insurmountable. The same paper presents a technique to convert some proofs in the random oracle model to a post-quantum scenario at the cost of having an additional cryptographic assumption. For signature schemes, if the proof in the random oracle model computes the response for the oracle queries on the fly, but computes all queries in the same way independent of previous events during the simulation and also avoids the techniques 1, 2 and 4 above described. Then, the proof can be adapted and is valid in a post-quantum setting.

In this dissertation, we use the random oracle model in some proofs, and we will point when these proofs are valid or not against quantum adversaries.

3 CHAMELEON HASH FUNCTIONS: DEFINITIONS AND PROPERTIES

This chapter is organized as follows:

- Section 3.1 presents the definition of chameleon hash functions and its security requirements.
- Section 3.2 presents basic properties of chameleon hash functions derived from its definitions
- Section 3.3 presents more constructions of chameleon hash functions given different assumptions
- Section 3.4 presents applications of chameleon hash functions

3.1 DEFINITION

Like we did in the previous chapter for other cryptographic primitives, here we will define the chameleon hash primitive.

A Chameleon Hash Scheme, proposed initially in (KRAWCZYK; RABIN, 1998), is a tuple of three algorithms: (CH.KeyGen, CH.Hash, CH.Collision) over sets (M, R, D) such that:

- CH.KeyGen is a probabilistic algorithm invoked as $(ek, tk) \xleftarrow{\$} CH.\text{KeyGen}(\lambda)$. Where ek is the **evaluation key** and tk is the **trapdoor key**. The input λ is the security parameter.
- CH.Hash is a deterministic algorithm invoked as $dgt \xleftarrow{\$} CH.\text{Hash}(ek, msg, rnd)$, where ek was returned by CH.KeyGen, $msg \in M$ is a **message**, $rnd \in R$ is a **randomness**, or a **random parameter** and $dgt \in D$ is a **digest**.
- CH.Collision is a probabilistic algorithm invoked as $rnd_2 \xleftarrow{\$} CH.\text{Collision}(tk, msg_1, rnd_1, msg_2)$ where $msg_1, msg_2 \in M$, $rnd_1, rnd_2 \in R$.
- We require that for all pairs (ek, tk) returned by CH.KeyGen, if rnd_2 was returned by $CH.\text{Collision}(tk, msg_1, rnd_1, msg_2)$, then $CH.\text{Hash}(ek, msg_1, rnd_1) = CH.\text{Hash}(ek, msg_2, rnd_2)$.
- The set R should be efficiently sampleable and recognizable using only information public in the evaluation key ek.

We consider a chameleon hash secure if it has two properties: uniformity and collision-resistance.

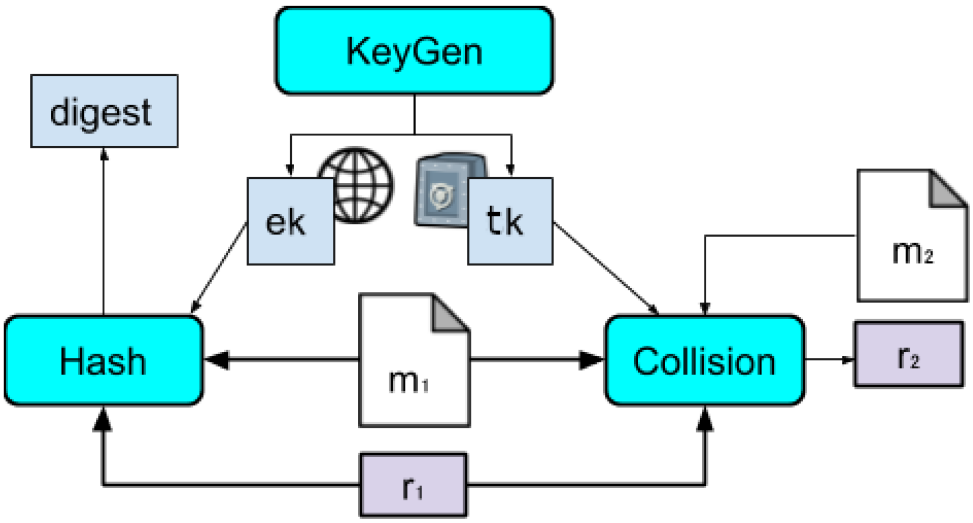

Figure 18 – Diagram representing the algorithms in a chameleon hash scheme. In this image, we have that $Hash(ek, m_1, r_1) = Hash(ek, m_2, r_2)$.

Definition 3 *A Chameleon Hash Scheme CH defined over* (M, R, D) *has the* ***uniformity property*** *if given any pair* (ek, tk) *returned by* $CH.\text{KeyGen}(\lambda)$*, for all pair of messages* $msg_1, msg_2 \in M$*, both functions* $f_1 = CH.\text{Hash}(ek, msg_1, \cdot)$ *and* $f_2 = CH.\text{Hash}(ek, msg_2, \cdot)$ *have the same probability distribution if the input is chosen uniformly at random from* R*.*

For example, Figure 19 illustrate two toy examples. It represents as graphs a chameleon hash over (M, R, D), with $M = \{m_1, m_2, m_3, m_4\}$, $R = \{r_1, r_2, r_3, r_4\}$ and $D = \{d_1, d_2, d_3\}$ and shows how the chameleon hash behaves for two different keys ek. The circles are messages and the squares are the possible digests. Each randomness r_i connect a given message m_i and digest d_i if $CH.\text{Hash}(ek, m_i, r_i) = d_i$. The left graph shows a distribution of a non-uniform chameleon hash. If we use the message m_3 we would get the digest d_1 with greater probability but if we choose the message m_2 we would get the digest d_2 with a greater probability. The graph in the right show what is expected in an uniform chameleon hash for all keys: even if the digest d_1 is obtained with greater probability, all messages produce each of the possible digests with the same probability.

The next property, **collision resistance**, is defined with the help of the following attack game:

Attack Game 14 ***(Collision Resistance).*** *For a given chameleon hash scheme CH defined over* (M, R, D)*, assume we have an adversary and a challenger, both initialized by the security parameter* λ*.*

The challenger computes a pair of keys (ek, tk) *using* $CH.\text{KeyGen}(\lambda)$ *and sends* ek *to the adversary.*

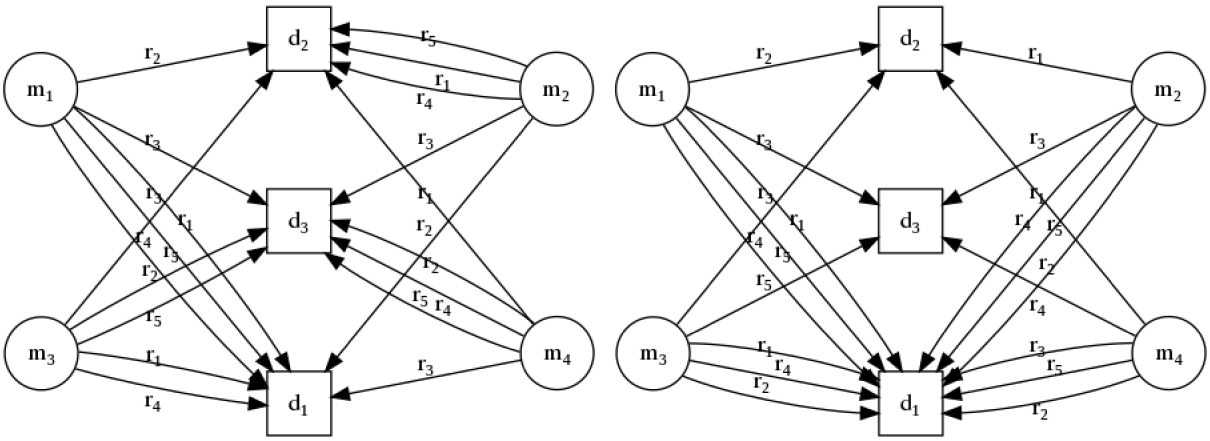

Figure 19 – Representations of non-uniform chameleon hash (left) and uniform chameleon hash (right).

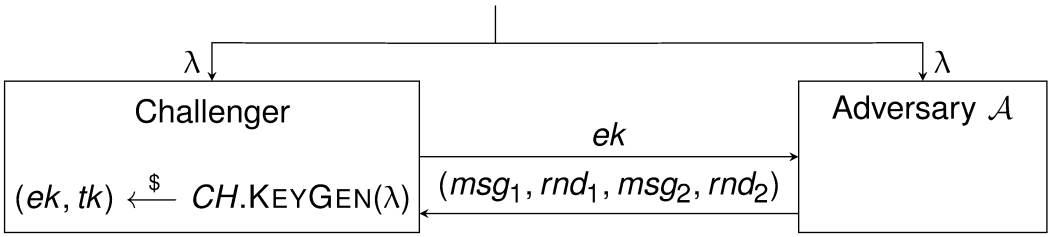

Figure 20 – Attack Game 1: Collision resistance

The adversary outputs a pair $((msg', rnd'), (msg'', rnd''))$*. We say that the adversary wins this game if* $(msg', rnd') \neq (msg'', rnd'')$ *and if* $CH.Hash(ek, msg_1, rnd_1) = CH.Hash(ek, msg_2, rnd_2)$*.*

We denote by $CRadv[\mathcal{A}, CH]$ *the probability of a given adversary* $\mathcal{A}$ *wins this game in respect to the chameleon hash CH.*

Notice that in the previous attack game, the challenger do not know the trapdoor key *tk*. Also the attacker does not have access to any known collision. Under these restrictions, we can define the second security requirement for chameleon hash functions:

Definition 4 *A chameleon hash scheme CH has* ***Collision Resistance*** *if for all efficient adversaries* $\mathcal{A}$*, the value of* $CRadv[\mathcal{A}, CH]$ *is negligible.*

3.2 PROPERTIES OF CHAMELEON HASH FUNCTIONS

By our definition, if we generate *ek* using the appropriate key generation algorithm, then $CH.\text{HASH}(ek, \cdot, \cdot)$ is a proper collision-resistant hash function than maps an input from $M \times R$ to D. Therefore, all theorems presented for hash functions in Subsection 2.3.2 can be adapted for chameleon hash functions. The attack games for second-preimage resistance and first preimage resistance can be adapted making

each challenger first compute keys (ek, tk), pass ek to the adversary and then proceed like in the definition for hash functions.

Theorem 4 *All collision-resistant chameleon hash functions also are second-preimage resistant.*

The proof for the above theorem is the same than what was presented for Theorem 1. Which means that for all adversaries $\mathcal{A}$ that can find the second-preimages in chameleon hash functions, we can build an adversary $\mathcal{B}$ that can bind collisions such that:

$$SPREadv[\mathcal{A}, CH] \leq CRadv[\mathcal{B}, CH]$$

Theorem 5 *All chameleon hash functions that are second-preimage resistant are also one-way functions.*

Proof: Here, instead of referring to the proof of Theorem 2, we can make a new proof specific for chameleon hash functions that achieve a better security bound.

Like in the previous proof, we define an adversary that given the security parameter λ, a chameleon hash evaluation key ek and some chameleon hash input $(msg, rnd) \in M \times R$ can compute the second preimage for the chameleon hash CH given an adversary $\mathcal{A}$ that can find the first preimage for the same chameleon hash CH:

- Adversary $\mathcal{B}(\lambda, ek, msg, rnd)$:
  1. $dgt \leftarrow CH.\text{HASH}(ek, msg, rnd)$
  2. $(msg', rnd') \xleftarrow{\$} \mathcal{A}(\lambda, ek, dgt)$
  3. **return** (msg', rnd')

This is almost the same adversary defined for the proof of Theorem 2, we just changed the adversaries to get an additional parameter ek and to take as input msg and rnd instead of a single message. This adapts that adversary to the security definitions of chameleon hash functions.

Contrary to Theorem 2, here we do not need to consider the possibility that $\mathcal{A}$ finds preimages only for digests that have a single preimage. In a chameleon hash, all digests have at least $|M|$ different preimages, or the algorithm $CH.\text{COLLISION}$ would be impossible.

Therefore, the above adversary $\mathcal{B}$ succeed at finding a second-preimage if, and only if adversary $\mathcal{A}$ succeeds and if $(msg, rnd) \neq (msg', rnd')$.

The probability of having $(msg, rnd) = (msg', rnd')$ is at most $1/|M|$ because this is the minimum number of inputs that maps for some digest and one of these inputs was

chosen uniformly at random. Then the probability of having if $(msg, rnd) = (msg', rnd')$ is at least $(1 - 1/|M|)$.

This means that the lower bound for the success of $\mathcal{B}$ is given by:

$$SPREadv[\mathcal{B}, CH] \geq (1 - \frac{1}{|M|})OWadv[\mathcal{A}, CH]$$

Rewriting this inequation, we conclude that for all adversaries $\mathcal{A}$ that can find the first preimages for the chameleon hash *CH*, then we can build an adversary $\mathcal{B}$ than can find second-preimages such that:

$$OWadv[\mathcal{A}, CH] \leq \frac{|M|}{|M|-1} SPREadv[\mathcal{B}, CH]$$

By our assumption, *CH* is second-preimage resistant. This means that the probability $SPREadv[\mathcal{B}, CH]$ is always negligible and in the above inequation is multiplied by a value polynomially bounded. This means that for any $\mathcal{A}$, the probability of finding a first preimage is negligible. ■

Which means that we can also use chameleon hash functions as one-way functions with negligible security loss.

Theorem 6 *The composition of a secure hash function and a collision-resistant chameleon hash is also a chameleon hash.*

More precisely, given a collision-resistant chameleon hash scheme CH defined over (M, R, D) *and a collision-resistant hash function* $HASH : M' \rightarrow M$ *we can define a composition* $CH \circ HASH$ *which is a chameleon hash defined over* (M', R, D)*. In this composition* $(CH \circ HASH).KeyGen = CH.KeyGen$*,* $(CH \circ HASH).Hash(ek, msg, rnd) = CH.Hash(ek, HASH(msg), rnd)$ *and* $(CH \circ HASH).Collision(ek, msg_1, rnd_1, msg_2) = CH.Collision(ek, HASH(msg_1), rnd, HASH(msg_2))$*.*

We can also use a collision-resistant chameleon-hash CH defined over (M, R, D) *and a hash function* $HASH : D \rightarrow D'$ *to define a new chameleon hash* $(HASH \circ CH)$ *over* (M, R, D')*. In this composition, we have* $(HASH \circ CH).KeyGen = CH.KeyGen$*,* $(HASH \circ CH).Hash(ek, msg, rnd) = HASH(CH.Hash(ek, msg, rnd))$ *and* $(HASH \circ CH).Collision = CH.Collision$*.*

Both compositions $CH \circ HASH$ *and* $HASH \circ CH$ *are collision-resistant if CH and HASH are also collision-resistant.*

Proof: In both cases, if an adversary is able to find collisions in the composition, it means that a collision is also found in *HASH* or in *CH*. This means that for all efficient adversaries $\mathcal{A}$ able to find collisions in the composition, exist adversaries $\mathcal{B}$ and $\mathcal{B}'$ able to find collisions respectively in *CH* and *HASH* such that:

$$CRadv[\mathcal{A}, CH \circ HASH] \leq CRadv(\mathcal{B}, CH) + CRadv(\mathcal{B}', HASH)$$

And also:

$$CRadv[\mathcal{A}, HASH \circ CH] \leq CRadv(\mathcal{B}, CH) + CRadv(\mathcal{B}', HASH)$$

Therefore, if both $CRadv[\mathcal{B}, CH]$ and $CRadv[\mathcal{B}', HASH]$ are negligible for all possible adversaries , this means the probability of finding collisions in the compositions are also negligible. ■

Notice that the uniformity property is also preserved: if the chameleon hash *CH* is uniform, both $CH \circ HASH$ and $HASH \circ CH$ are uniform.

3.3 CONSTRUCTIONS OF CHAMELEON HASH FUNCTIONS

Now we will present some possible constructions of the chameleon hash scheme using some assumptions from Chapter 2: the hardness of the discrete logarithm problem and the existance of one-way functions.

3.3.1 Chameleon Hash from Discrete Logarithm Assumption

This chameleon hash (denoted CH_{DL}) is presented as such in (KRAWCZYK; RABIN, 1998), but the technique was already known in the form of a chameleon commitment scheme.

This construction uses the discrete logarithm assumption, and to define it, we will use the same algorithm GENGROUP specified in Subsection 2.2.2 to generate a multiplicative group with prime order.

Let G be a multiplicative group with order q. The chameleon hash CH_{DL} is defined over sets (M, R, D) where $M = R = \mathbb{Z}_q$ and $D = G$. The scheme algorithms are defined as:

- $CH_{DL}.KeyGen(\lambda)$:
  1. $(G, q, g) \xleftarrow{\$} \text{GENGROUP}(\lambda)$
  2. $x \xleftarrow{\$} \mathbb{Z}_q^+$
  3. $y \leftarrow g^x$
  4. $ek \leftarrow (G, q, g, y)$
  5. $tk \leftarrow (G, q, g, x)$
  6. **return** (ek, tk)
- $CH_{DL}.\text{HASH}(ek, m, r)$:
  1. $(G, q, g, y) \leftarrow ek$
  2. **return** $g^m y^r$

- CH_{DL}.COLLISION(tk, m_1, r_1, m_2):
  1. $(G, q, g, x) \leftarrow tk$
  2. **return** $(m_1 + xr_1 - m_2)x^{-1}$

The above construction works because if $r_2 \leftarrow CH_{DL}$.COLLISION(tk, m_1, r_1, m_2), then:

$$\begin{aligned} CH.\text{HASH}(ek, m_2, r_2) &= g^{m_2} y^{(m_1+xr_1-m_2)x^{-1}} \\ &= g^{m_2} g^{x(m_1+xr_1-m_2)x^{-1}} \\ &= g^{m_2} g^{(m_1+xr_1-m_2)} \\ &= g^{m_2+m_1+xr_1-m_2} \\ &= g^{m_1} y^{r_1} = CH.\text{HASH}(ek, m_1, r_1) \end{aligned}$$

Theorem 7 *The chameleon hash CH_{DL} has the uniformity property.*

Proof: For any $m, r \in \mathbb{Z}_q$, the algorithm CH_{DL}.HASH produces a result computing g^{m+rx}. If the randomness r is chosen uniformly at random, then the exponent will also be a random and uniform value in $\mathbb{Z}_q$, no matter what are the values of m and x, provided that $x \neq 0$. Therefore, given a random and uniform r we will not be able to distinguish between the hash of different messages $m \in M$. ■

Theorem 8 *The above chameleon hash CH_{DL} is collision-resistant if the discrete logarithm assumption is true for $\mathbb{Z}_q$.*

Proof: We will show that if we have an adversary $\mathcal{A}$ able to find collisions in this chameleon hash, then we can build an adversary $\mathcal{B}$ to solve the discrete logarithm problem described in Section 2.2.2. Our adversary $\mathcal{B}$ is composed of the following parts:

- **Initialization:** Adversary $\mathcal{B}$ is initialized by the security parameter λ. It also takes as input from its challenger the tuple (Q, q, g, y) as defined in the discrete logarithm attack game. It produces the chameleon hash evaluation key $ek = (G, q, g, y)$ directly from this input. Next, it initializes the chameleon hash collision-finder $\mathcal{A}$ with the security parameter λ and with the key ek.
- **Finalization:** In the end the adversary $\mathcal{A}$ outputs a possible collision $(m', r'), (m'', r'')$. Assuming that this is really a collision for CH_{DL}, adversary $\mathcal{B}$ finds and returns the solution $x' = (m'' - m')(r' - r'')^{-1}$.

The above adversary $\mathcal{B}$ works because given any arbitrary collision for CH_{DL}, it is possible to extract the value x (and with this, it is possible to recreate the trapdoor tk). Because if $g^{m'}y^{r'} = g^{m''}y^{r''}$, then this means that $g^{m'+xr'} = g^{m''+xr''}$ and $m' + xr' = m'' + xr''$. Any revealed collision in the chameleon hash exposes the value x and the chameleon hash trapdoor.

This means that for all adversaries $\mathcal{A}$ that can find collisions in CH_{DL}, we can build an adversary $\mathcal{B}$ that wins the discrete logarithm attack game such that:

$$CRadv[\mathcal{A}, CH_{DL}] \leq DLadv[\mathcal{B}, G]$$

Therefore, if computing the discrete logarithm for G is hard, the right side of this inequation is always negligible. Which means that CH_{DL} is collision-resistant. ■

The above security proof shows the collision-resistance for CH_{DL}, but also exposes a severe fragility that limits the usage of most kinds of chameleon hash functions. Using a trapdoor, we indeed can find collisions for the CH.HASH algorithm. However, we should not expose or show these collision for other people, or the trapdoor would be exposed and the collision-resistance cannot be assumed anymore.

3.3.2 Chameleon Hash from Homomorphic One-Way Functions

This construction was presented first at (ALAMATI et al., 2019). The article tried to build different cryptographic primitives using only one-way functions given additional properties. For chameleon hash functions, it proposed that they could be built using an one-way function with homomorphic properties. In this work we add that this homomorphic one-way function must also be collision-resistant and show this in Theorem 10.

To build the chameleon hash scheme, we need two groups $(X, \oplus)$ and $(Y, \otimes)$ and also a collision-resistant one-way function $OW : X \to Y$, such that:

$$OW\left(\bigoplus_{i\in[1,k]} x_i\right) = \bigotimes_{i\in[1,k]} OW(x_i)$$

The chameleon hash CH_{OW} is defined over (M, X, Y) with $M = \{0,1\}^k$ for some constant k. The scheme's algorithms are defined as:

- CH_{OW}.KEYGEN:
  1. $\mathbf{T} \xleftarrow{\$} X^{2\times k}$
  2. **for** $i \in \{1, \ldots k\}$:
  3. **for** $j \in \{1, 2\}$:
  4. $\mathbf{E}[i][j] \leftarrow OW_\lambda(\mathbf{T}[i][j])$

5. **return** (ek = **E**, tk = **T**)

Both the trapdoor and the evaluation key are a matrix with 2 lines and k columns. The first is chosen uniformly at random from X and the second is obtaining applying the one-way function OW_λ to each element in the trapdoor.

The other two algorithms, CH_{OW}.HASH and CH_{OW}.COLLISION are defined as below:

- CH_{OW}.HASH:

$$CH_{OW}.\text{HASH}(ek = \mathbf{E}, \mathbf{m}, x) = \left(\bigotimes_{i \in [1,k]} \mathbf{E}[i][\mathbf{m}[i] + 1] \right) \otimes OW_\lambda(x)$$

- CH_{OW}.COLLISION:

$$CH_{OW}.\text{COLLISION}(tk = \mathbf{T}, \mathbf{m}, x, \mathbf{m'}) = \left(\bigoplus_{i \in [1,k]} \mathbf{T}[i][\mathbf{m'}[i] + 1] \right)^{-1} \oplus \left(\bigoplus_{i \in [1,k]} \mathbf{T}[i][\mathbf{m}[i] + 1] \right) \oplus x$$

We choose the elements used in the computation in **E** and **T** depending if each bit in **m** is 0 or 1. If the element is 0, we use the first line of the matrix, if the element is 1, we use the second line. As we number the lines in the matrices starting with 1, in the formulas above we add 1 to each bit to get the correct line.

This constructions works because if we apply the function OW_λ to a randomness x' returned by CH_{OW}.COLLISION(tk, **m**, x, **m'**), we get:

$$\left(\bigotimes_{i \in [1,k]} OW_\lambda(\mathbf{E}[i][\mathbf{m'}[i] + 1]) \right)^{-1} \otimes \left(\bigotimes_{i \in [1,k]} OW_\lambda(\mathbf{E}[i][\mathbf{m}[i] + 1]) \right) \oplus OW_\lambda(x)$$

Computing a chameleon hash using **m'** and the x' is equivalent to using the operator $\otimes$ between the above value and $\bigotimes_{i \in [1,k]} OW_\lambda(\mathbf{E}[i][\mathbf{m'}[i] + 1])$. Canceling the inverses, the result becomes equal the chameleon hash CH_{OW}.HASH(ek, **m**, x) as expected.

Theorem 9 *The chameleon hash CH_{OW} has the uniformity property.*

Given any two messages $\mathbf{m}_1$ and $\mathbf{m}_2$ and given random and uniform x_1 and x_2, we have:

$$x'_1 = \left(\bigoplus_{i \in k} \mathbf{T}[i][\mathbf{m}_1[i] + 1] \right) \oplus x_1 \qquad x'_2 = \left(\bigoplus_{i \in k} \mathbf{T}[i][\mathbf{m}_2[i] + 1] \right) \oplus x_2$$

Independent of the two messages and the keys, both x'_1 and x'_2 are random and uniform provided that x_1 and x_2 are also random and uniform.

By definition:

$$CH_{OW}.\text{HASH}(ek, \mathbf{m}_1, x_1) = OW(x'_1) \qquad CH_{OW}.\text{HASH}(ek, \mathbf{m}_2, x_2) = OW(x'_2)$$

And so the probability distribution of two different messages are the same if the random parameter is chosen random and uniform from X. ■

Theorem 10 *If OW is not collision-resistant family of one-way functions, then the chameleon hash CH_{OW} also is not collision-resistant.*

Proof: If *OW* is not collision-resistant, this means that exist an efficient adversary $\mathcal{A}$ that receives a security parameter λ and returns a pair of collisions for OW_λ: $(x', x'') \xleftarrow{\$} \mathcal{A}(\lambda)$.

We can use this adversary to build a collision-finder $\mathcal{B}$ for CH_{OW}:

- $\mathcal{B}(\lambda, ek)$:
  1. $\mathbf{m} \xleftarrow{\$} \{0,1\}^k$
  2. $(x', x'') \xleftarrow{\$} \mathcal{A}(\lambda)$
  3. **return** $(\mathbf{m}, x'), (\mathbf{m}, x'')$

The above collision-finder finds a collision for CH_{OW} with the same probability than $\mathcal{A}$ succeeds finding collisions in *OW*. Therefore, it succeeds with non-negligible probability and this means that CH_{OW} is not collision-resistant.

■

If we have as OW_λ a permutation function, then there are no possible collisions and the chameleon hash would not be susceptible to such attack.

Theorem 11 *If OW is a family of collision-resistant one-way homomorphic functions, the chameleon hash scheme CH_{OW} is collision-resistant.*

For all adversaries $\mathcal{A}$ which find collisions in this chameleon hash scheme, we can build a new adversary $\mathcal{B}$ to find a preimages in *OW*. The adversary $\mathcal{B}$ can be constructed as:

- $\mathcal{B}(\lambda, y)$:
  1. $(ek = \mathbf{E}, tk = \mathbf{T}) \xleftarrow{\$} CH_{OW}.\text{KEYGEN}(\lambda)$
  2. $index \xleftarrow{\$} \{1, \ldots, k\}$

3. $b \xleftarrow{\$} \{1,2\}$
4. $\mathbf{E}[index][b] \leftarrow y$
5. $((\mathbf{m'}, x'), (\mathbf{m''}, x'')) \leftarrow \mathcal{A}(ek)$
6. **if** $\mathbf{m'}[index] \neq \mathbf{m''}[index]$:
7. **if** $\mathbf{m'}[index] = b$:
8. $\mathbf{m}_1 \leftarrow \mathbf{m'}$; $x_1 \leftarrow x'$; $\mathbf{m}_2 \leftarrow \mathbf{m''}$; $x_2 \leftarrow x''$
9. **else if** $\mathbf{m''}[index] = b$:
10. $\mathbf{m}_1 \leftarrow \mathbf{m''}$;$x_1 \leftarrow x''$;$\mathbf{m}_2 \leftarrow \mathbf{m'}$;$x_2 \leftarrow x'$
11. $x' \leftarrow \left(\bigoplus_{i\in[1,index-1]} \mathbf{T}[i][\mathbf{m}_1[i]+1]\right)^{-1} \oplus \left(\bigoplus_{i\in[index+1,r]} \mathbf{T}[i][\mathbf{m}_1[i]+1]\right)^{-1}$
12. $x' \leftarrow x' \oplus x_1^{-1}$
13. $x' \leftarrow x' \oplus \left(\bigoplus_{i\in[1,k]} \mathbf{T}[i][\mathbf{m}_2[i]+1]\right) \oplus x_2$
14. **return** x'

The above adversary $\mathcal{B}$ works producing a valid pair of keys for the chameleon hash, but it changes one random element in the matrix **E** to the value y whose preimage it wants to discover. Note that in the specific random position where the element y was inserted in matrix **E** at line 4, the matrix **T** does not have the right corresponding preimage for y as we would expect from a valid key tk. However, the trapdoor tk is never sent to adversary $\mathcal{A}$ and so, our adversary $\mathcal{B}$ is correctly simulating a challenger for $\mathcal{A}$.

Then, assuming that $\mathcal{A}$ succeed at finding a collision, adversary $\mathcal{B}$ checks if both messages returned have different values at the random position corresponding to where it stored y in the matrix **E**. If this is true, then the preimage of y in OW_λ can be deduced. It is the value that should be in the matrix **T** in the same position where y was placed in **E**. Knowing a collision where only one of the messages in the pair (**m'**, **m"**) needs to evaluate y and knowing all other preimages involved in the computation, adversary $\mathcal{B}$ use the definition of CH_{OW}.COLLISION to create an equation where the preimage of y is the only unknown. Then, solving this equation yields this preimage in lines 11, 12 and 13.

Three conditions need to be satisfied to the above adversary $\mathcal{B}$ succeed at finding a preimage for a given y in OW_λ:

1. The adversary $\mathcal{A}$ must succeed at finding a collision in line 5. The probability of this happening is $CRadv[\mathcal{A}, CH_{OW}]$.

2. The collision returned by $\mathcal{A}$ in line 5 must have $msg^* \neq msg^{**}$. If this is not true, but $\mathcal{A}$ really found a collision, then this means that $x' \neq x''$ and $OW_\lambda(x') = OW_\lambda(x'')$.

Therefore, the probability of this not happening can be modeled by the probability of some adversary $\mathcal{B}'$ finding a collision in OW_λ: $CRadv[\mathcal{B}', OW]$.

3. Given the random index chosen at line 2, it must be true that **m'**[*index*] $\neq$ **m"**[*index*]. Otherwise, $\mathcal{B}$ would not be able to deduce an equation with a single unknown representing the preimage of y. This is the condition tested in line 6.

If in line 5 adversary $\mathcal{A}$ succeed at finding a collision, then or it also found a collision in OW_λ (when **m'** = **m"** and the second condition above is not satisfied) or it found a useful collision that $\mathcal{B}$ can use to find a preimage (when **m'** $\neq$ **m"** satisfying the second condition above). Therefore, the probability of finding an useful collision can be modeled by $CRadv[\mathcal{A}, CH_{OW}] - CRadv[\mathcal{B}', OW]$

If $\mathcal{A}$ produced an useful collision, this means that at least one of the elements in the vectors **m'** and **m"** are different. Then, the probability of having **m'**[*index*] $\neq$ **m"**[*index*] is at least $1/k$.

Combining these results, we have that the lower bound for the success of $\mathcal{B}$ is given by:

$$OWadv[\mathcal{B}, OW] \geq \frac{1}{k}(CRadv[\mathcal{A}, CH_{OW}] - CRadv[\mathcal{B}', OW])$$

Rewriting the above inequation, we have that for all adversaries $\mathcal{A}$, we can build adversaries $\mathcal{B}$ and $\mathcal{B}'$ such that:

$$CRadv[\mathcal{A}, CH] \leq k \cdot OWadv[\mathcal{B}, OW] + CRadv[\mathcal{B}', OW]$$

By our hypothesis, we know that OW is one-way and also collision-resistant. Therefore, the right side of this inequation is negligible. Then $CRadv[\mathcal{A}, CH]$ must also be negligible and the chameleon hash is collision-resistant. ■

3.3.3 Other Constructions

In (SHAMIR; TAUMAN, 2001) was proposed another chameleon hash over (M, R, D) based on factoring assumption. However, that construction does not have the required property of having set R efficiently recognizable and sampleable without sacrificing the uniformity property. To produce a valid randomness $r \in R$ it was necessary to have sensible information from the trapdoor key tk. That information could not be placed in the evaluation key ek without compromising the collision resistance of the construction.

In (ATENIESE; MEDEIROS, 2004) there are a list of different chameleon hash functions defined using a generalization of the scheme presented here, where the CH.HASH and CH.COLLISION use as additional parameter a tag that identify each use of the chameleon hash. However, all constructions in that paper have its proofs based on heuristics like the random oracle model and the generic model. The same applies

some other tag-based and identity-based chameleon hash functions in the literature, like the construction from (ZHANG, F. et al., 2003).

3.4 APPLICATIONS

Here we list some applications for chameleon hash functions proposed in the literature.

3.4.1 Chameleon Signatures

Usually a signature scheme has the following properties:

- The signer cannot repudiate a signed document if the private key is not compromised. Only the signer knows the private key and only him can create a valid signature.
- The recipient can show the document and signature to any person and this proves that the document was originated from the signer and signed with the associated private key to the public key.

Sometimes the second property can be a problem, as it conflicts with the signer's privacy. In some applications it can be desirable to keep the signature verifiable only for some users, but if they show the message and the signature, they cannot convince third parties that the message was really signed by the signer. We can achieve this by the following method using any signature scheme *SIG* and any chameleon hash scheme *CH*:

1. The recipient Bob runs $(ek, tk) \leftarrow CH.\text{KEYGEN}(\lambda)$ to get the keys for a chameleon hash scheme.
2. The signer Alice runs $(pk, sk) \leftarrow SIG.\text{KEYGEN}(\lambda)$ to get the keys for a signature scheme.
3. Alice and Bob exchange keys. Bob sends *ek* and Alice sends *pk*.
4. To sign a message *msg*, Alice choose a random *rnd* and runs the algorithm $SIG.\text{SIGN}(sk, CH.\text{HASH}(ek, msg, rnd)||rnd)$ to obtain the signature *sig*. She sends (msg, rnd, sig) to Bob.
5. Bob can verify the signature with $SIG.\text{VERIFY}(pk, CH.\text{HASH}(ek, msg, rnd), sig)$.

If Bob got a signed message from Alice and the signature was valid, Alice cannot repudiate the signed message, as only she knows the signature secret key to produce

a valid signature, and only Bob knows the trapdoor key to compute collisions in the chameleon hash.

However, if Bob tries to show the message and signature to other people, he cannot prove for anyone that the signature is valid. As Bob knows the trapdoor key, he can compute collisions in the chameleon hash and impute any message to Alice. The only disclosed information for third parties if Bob shows a signed message is that at some time Alice sent a signed message to Bob, but the real content is unknown.

If necessary in some contexts, Alice can prove that she did not sign some message. For example, if Bob falsely accuses her of confessing a crime in a message, she can proof that she did not sign that message showing another signed message with the same hash. As she does not know the trapdoor tk, providing a collision shows that the first message really was not signed by Alice.

However, this presents a possible attack against this scheme: Alice can use this chameleon signature to tell Bob some secret. Bob can then force her to reveal this secret to the world forging a false message with a serious crime confession and attaching her signature to the message. Now to prove her innocence in a very serious crime, Alice must reveal a collision for the chameleon hash CH. However, as she do not know the trapdoor tk, she can do so only revealing the original signed message with her secret.

This problem can be avoided using some specific constructions of chameleon hash functions instead of using the chameleon hash as a black-box. Using CH_{DL}, for example, Alice can use the following algorithm for producing a new collision in the chameleon hash, provided that she has the original message (msg, rnd) and a false message (msg', rnd') that are a collision in this scheme:

- THIRDCOLLISION(msg, rnd, msg', rnd'):
  1. $msg'' \xleftarrow{\$} M$
  2. $x \leftarrow (msg' - msg)(rnd - rnd')^{-1}$
  3. $rnd'' \leftarrow (msg - msg'' + rnd)x^{-1}$
  4. **return** (msg'', rnd'')

The above algorithm works because as discussed in the security proof for CH_{DL}, if someone discovers a collision in CH_{DL} using the evaluation key ek, is trivial to deduce the corresponding trapdoor key tk. With the appropriate trapdoor, new collisions can be obtained.

This also means that such chameleon signatures should be one-time signatures. Otherwise, the possibility of repudiate a signed message is weakened: if compute a collision in the chameleon hash exposes the trapdoor, the recipient Bob is less likely to forge messages risking the secret of his trapdoor.

3.4.2 On-line/Off-line Signatures

This application was first presented at (SHAMIR; TAUMAN, 2001).

Notice that in the chameleon hash CH_{DL}, computing collisions is usually faster than computing hash functions. In the first case we need to compute two multiplications, two sums and the inverse of an element. In the second case we need to compute a multiplication and two exponentiations, which usually is a slower operation.

Given a chameleon hash *CH* with this property, and given a signature scheme *SIG*, we can create a new signature scheme $SIG_{On/Off}$ that supports on-line/off-line signatures.

The idea is that in the signature scheme $SIG_{On/Off}$, even when we still do not know what will be the next signed message, we can compute in our spare time part of the future signature. When we finally discover what will be the next signed message, we resume the computation that we started before and thus the signature appears to be computed faster. We want to be able to compute most part of the signature before knowing the signed message.

We can model this as a signature scheme with four different algorithms. The algorithms $SIG_{On/Off}$.KEYGEN and $SIG_{On/Off}$.VERIFY works as before. However, instead of a single signing algorithm, we have two:

- $SIG_{On/Off}$.OFFLINESIGN(sk) $\rightarrow$ *presig*
- $SIG_{On/Off}$.ONLINESIGN(sk, *presig*, *msg*) $\rightarrow$ *sig*

The first algorithm can be precomputed much before we know the signed message and the result, *presign* is then stored. When we know the message to be signed, we recover the stored *presig* and compute the real signature *sig*. These two algorithms can be used to define a more traditional signing algorithm as:

- $SIG_{On/Off}$.SIGN(sk, *msg*):
  1. *presig* $\xleftarrow{\$}$ $SIG_{On/Off}$.OFFLINESIGN(sk)
  2. *sig* $\xleftarrow{\$}$ $SIG_{On/Off}$.ONLINESIGN(sk, *presig*, *msg*)
  3. **return** *sig*

With a chameleon hash *CH* where computing collisions is faster than computing hash functions defined over (M, R, D) and with any signature scheme defined over (D, S), we can build an online/offline signature scheme $SIG_{On/Off}$ defined over (M, $S \times R$) with the following algorithms:

- $SIG_{On/Off}$.KEYGEN(λ):
  1. (pk, sk) $\xleftarrow{\$}$ *SIG*.KEYGEN(λ)

2. $(ek, tk) \xleftarrow{\$} CH.\text{KEYGEN}(\lambda)$
3. $pk' \leftarrow (pk, ek)$
4. $sk' \leftarrow (sk, ek, tk)$

- $SIG_{On/Off}.\text{OFFLINESIGN}(sk')$:
  1. $(sk, ek, tk) \leftarrow sk'$
  2. $msg' \xleftarrow{\$} M$
  3. $rnd' \xleftarrow{\$} R$
  4. $dgt \leftarrow CH.\text{HASH}(ek, msg', rnd')$
  5. $sig' \xleftarrow{\$} SIG.\text{SIGN}(sk, dgt)$
  6. $presig \leftarrow (sig', msg', rnd')$
  7. **return** *presig*
- $SIG_{On/Off}.\text{ONLINESIGN}(sk', presig, msg)$:
  1. $(sig', msg', rnd') \leftarrow presig$
  2. $(sk, ek, tk) \leftarrow sk'$
  3. $rnd \xleftarrow{\$} CH.\text{COLLISION}(tk, msg', rnd', msg)$
  4. $sig \leftarrow (sig', rnd)$
- $SIG_{On/Off}.\text{VERIFY}(pk', msg, sig)$:
  1. $(pk, ek) \leftarrow pk'$
  2. $(sig', rnd) \leftarrow sig$
  3. $dgt \leftarrow CH.\text{HASH}(ek, msg, rnd)$
  4. **return** SIG.VERIFY(pk, dgt, sig')

Notice that the computed collision never is leaked, so the chameleon hash remains secure, even after multiple signatures. Contrary to chameleon signatures, here the signer uses its own keys in the chameleon hash, not the recipient's keys. Therefore, the construction has the non-repudiability of a regular signature scheme.

3.4.3 Transforming GCMA-secure signatures in CMA-secure signatures

Combining a chameleon hash with a signature scheme also provides the additional property that the signature can become more secure after the combination. Assume that we have a signature scheme *SIG* which is secure against generic chosen

message attacks, as described in Subsection 2.3.5.1. We will show that we can combine this signature with any chameleon hash *CH* producing a new signature scheme secure against chosen-message attacks.

We will denote this new signature as $(SIG \circ CH)$ and its algorithms are defined as:

- $(SIG \circ CH).\text{KEYGEN}(\lambda)$:
  1. $(pk, sk) \xleftarrow{\$} SIG.\text{KEYGEN}(\lambda)$
  2. $(ek, tk) \xleftarrow{\$} CH.\text{KEYGEN}(\lambda)$
  3. $pk' \leftarrow (pk, ek)$
  4. $sk' \leftarrow (sk, ek)$
- $(SIG \circ CH).\text{SIGN}(sk', msg)$:
  1. $(sk, ek) \leftarrow sk'$
  2. $rnd \xleftarrow{\$} R$
  3. $dgt \leftarrow CH.\text{HASH}(ek, msg, rnd)$
  4. $sig' \xleftarrow{\$} SIG.\text{SIGN}(sk, dgt)$
  5. $sig \leftarrow (sig', rnd)$
  6. **return** *sig*
- $(SIG \circ CH).\text{VERIFY}(pk', msg, sig)$:
  1. $(pk, ek) \leftarrow pk'$
  2. $(sig', rnd) \leftarrow sig$
  3. $dgt \leftarrow CH.\text{HASH}(ek, msg, rnd)$
  4. **return** SIG.VERIFY(pk, dgt, sig')

Notice that we never use the trapdoor key *tk* in this construction. We use the chameleon hash as a regular hash function in the hash-and-sign paradigm. However, with chameleon hash functions we can prove that this construction is secure against adaptive chosen message attacks if *SIG* is secure against generic chosen message attacks.

Theorem 12 *Let SIG be a signature scheme defined over* (D, S) *secure against generic chosen message attacks and CH be a collision-resistant chameleon hash defined over* (M, R, D)*. Then, the signature scheme* $(SIG \circ CH)$ *over* $(M, S \times R)$ *is secure against chosen-message attacks.*

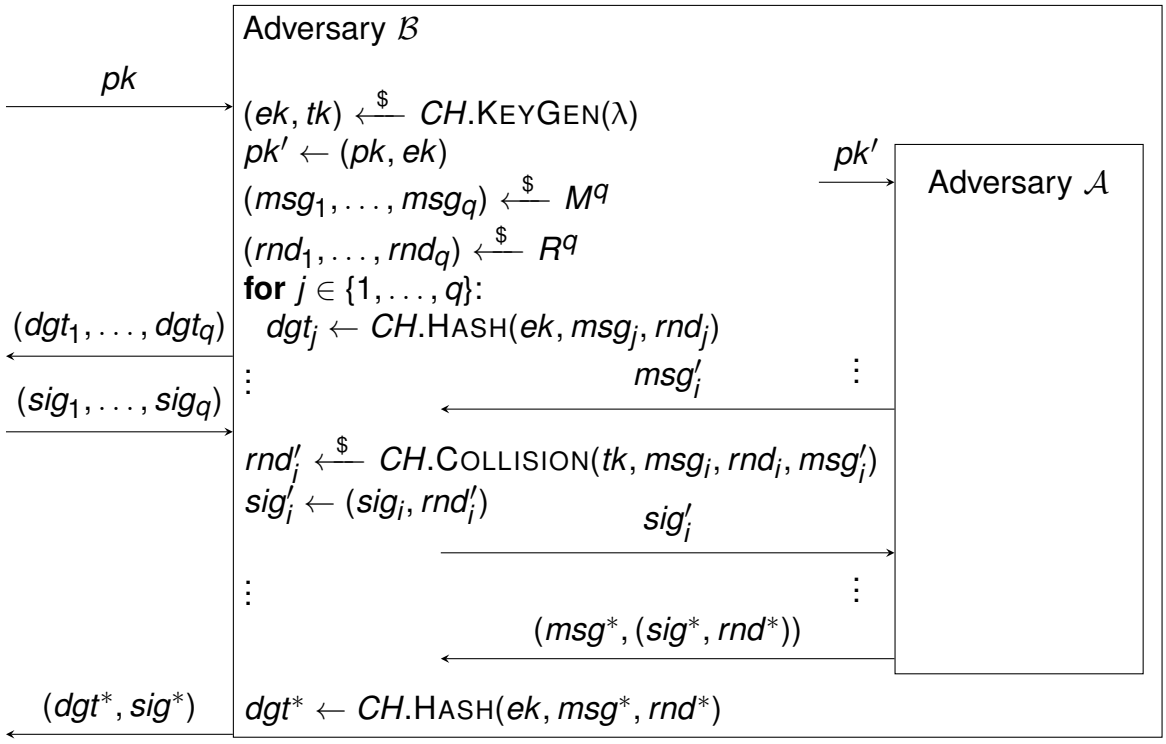

Figure 21 – Using adversary $\mathcal{A}$ that breaks the security of $(SIG \circ CH)$ to build adversary $\mathcal{B}$ that breaks the security of SIG

Proof: We will show how given an adversary $\mathcal{A}$, which attacks $(SIG \circ CH)$ with adaptive chosen message attacks, one could build an adversary $\mathcal{B}$ which attacks SIG using generic chosen message attacks. The full construction is illustrated in Figure 21.

Our forger $\mathcal{B}$ that interacts with generic-chosen message attack game, is composed of the following parts:

- **Initialization:** Here $\mathcal{B}$ takes as input pk, a public key for SIG. It proceeds to run $(ek, tk) \xleftarrow{\$} CH.\text{KEYGEN}(\lambda)$. Then it chooses a polynomial number of q random messages $(msg_1, \ldots, msg_q)$ and chooses uniformly at random the same number of values from R: $(rnd_1, \ldots, rnd_q)$. Finally, it computes $(dgt_1, \ldots, dgt_q)$ where all the elements of this tuple are computed as $dgt_i \leftarrow CH.\text{HASH}(ek, msg_i, rnd_i)$. It passes this tuple of digests to its challenger and gets as response a tuple $(sig_1, \ldots, sig_q)$ with the signature of each digest using SIG. Finally, it initializes the adversary $\mathcal{A}$ passing $pk' = (pk, ek)$ as its public key.

- **Queries:** The adversary $\mathcal{A}$ will do a polynomial number of queries. In each of them, it will send a message msg'_i. The adversary $\mathcal{B}$ produces a response computing $rnd'_i \leftarrow CH.\text{COLLISION}(tk, msg_i, rnd_i, msg'_i)$ and then replying with the valid signature (sig_i, rnd'_i).

- **Finalization:** After all the queries, the adversary $\mathcal{A}$ produces as forgery the tuple $(msg^*, (sig^*, rnd^*))$. First the adversary $\mathcal{B}$ checks if the forgery was successful. If

not, it halts. If yes, it produces its own forgery for signature scheme *SIG* sending (CH.HASH(ek, msg^*, rnd^*), sig^*) to its challenger.

The adversary $\mathcal{B}$ succeeds at producing a forgery if the following two events happen:

1. The adversary $\mathcal{A}$ succeeds when producing a forgery. This happens with probability $CMAadv[\mathcal{A}, (SIG \circ CH)]$.
2. The output for $\mathcal{A}$ is such that CH.HASH(ek, msg^*, rnd^*) is different than all the digests in the tuple $(dgt_1, \ldots, dgt_q)$. If this do not happen, this means that we found a collision in the chameleon hash CH. The probability of this happening is at most $CRadv[\mathcal{B}', CH]$ for some adversary $\mathcal{B}'$.

These two events are not independent. Adversary $\mathcal{A}$ in the worst case could succeed at creating forgeries precisely computing collisions in the chameleon hash and finding inputs where collisions are more probable. When $\mathcal{A}$ succeed in the above interaction, either we obtain an useless forgery that reveals a collision in the chameleon hash or we find a useful forgery that can be used to produce a forgery in *SIG*. We can model the probability of finding a useful forgery by $CMAadv[\mathcal{A}, SIG \circ CH] - CRadv[\mathcal{B}', CH]$ where $\mathcal{B}'$ is an adversary that finds collisions in CH simulating the above interaction between a challenger, adversary $\mathcal{A}$ and adversary $\mathcal{B}$.

The lower bond for the success of $\mathcal{B}$ is given by:

$$GCMAadv[\mathcal{B}, SIG] \geq CMAadv[\mathcal{A}, SIG \circ CH] - CRadv[\mathcal{B}', CH]$$

This means that for all efficient adversaries $\mathcal{A}$, one could build efficient adversaries $\mathcal{B}$ and $\mathcal{B}'$ such that:

$$CMAadv[\mathcal{A}, SIG \circ CH] \leq GCMAadv[\mathcal{B}, SIG] + CRadv[\mathcal{B}', CH]$$

By our hypothesis, the chameleon hash CH is collision-resistant and *SIG* is secure against generic chosen message attacks. Therefore, the right side of the above inequation is negligible and the probability of any adversary creating a forgery against $SIG \circ CH$ is a adaptive chosen message attack is also negligible.

■

This construction of a CMA-secure signature from a GCMA-secure one was first described at (SHAMIR; TAUMAN, 2001). However, the article (GENNARO et al., 1999) used this technique before, treating it as an ad-hoc method to signature scheme that could be proven secure against adaptive chosen message attacks without using the random oracle model.

3.4.4 Redactable Signatures

Partitioning a document in multiple parts and signing some parts with a standard signature and other parts with chameleon signatures, it is possible to create redactable signatures. With this technique a person can sign a document, but allow an authorized censor to remove, redact or replace specific parts of a document without invalidating the signature. This can be useful to anonimize sensible information in documents before releasing to the public and to ensure privacy for people mentioned in a document.

This application was presented first at (ATENIESE et al., 2005). And a more rigorous analysis for the security requeriments for such signatures is presented in (BRZUSKA et al., 2009).

The great difficulty in using chameleon hash functions to build redactable signatures is that it is difficult to maintain the security of chameleon hashes after a collision is revealed to attackers. In redactable signatures, having the original document and a redacted copy, or having two different redacted copies will expose a collision. Because of this, not all chameleon hashes are suitable to build redactable signatures.

3.4.5 Chameleon Hash Chains and Authentication

A hash chain is a sequence of values generated applying a hash function successively to the same input. Given some input msg and some function $HASH : M \rightarrow D$ with $D \subset M$, we can define the hash chain $(msg, HASH(msg), HASH(HASH(msg)), \ldots)$.

In (LAMPORT, 1981) hash chains were suggested as a form of authentication. A user could compute a hash chain $(msg, HASH(msg), \ldots HASH^n(msg))$ and send the last computed value to a server. Each time the user wants to authenticate, it can send a preimage of the last sent value to the server. This could be used to authenticate n times in a given server.

In fact, any One-Way function could be used in this authentication scheme, not necessarily collision-resistant hash functions.

Based on the concept of hash chains, chameleon hash chains also can be used for authentication. For this, we would need a chameleon hash CH defined over (M, R, D) with $D \subseteq M$. A server or some service can allow users to authenticate first requiring a registration, and then performing the verification step in each authentication.

In the registration step, the user needs to generate a pair of keys (ek, tk) using $CH.\text{KEYGEN}(\lambda)$ and choose randomly some $msg_0 \in M$ and $rnd_0 \in R$ to compute dgt_0 as $CH.\text{HASH}(ek, msg_0, rnd_0)$. The tuple (ek, dgt_0) is sent to the server and (msg_0, rnd_0) is stored.

In the verification step, in the first authentication we set $i = 1$ and for all other authentications we increment this value. The user choose uniformly at random new values $msg_i \in M$ and $rnd_i \in R$ and compute $auth_i$ ad $CH.\text{HASH}(ek, msg_i, rnd_i)$. Then,

it computes rnd'_i as $CH.\text{COLLISION}(tk, msg_{i-1}, rnd_{i-1}, auth_i)$. It sends to the server the pair $(auth_i, rnd'_i)$. The server authenticates the user only if $CH.\text{HASH}(ek, auth_i, rnd'_i) = dgt_{i-1}$. If so, the server also updates $dgt_i \leftarrow auth_i$ for the next authentication.

Notice that in each new authentication we produce a new digest that will be the target digest for the next authentication and also will be the message for the current authentication. The trapdoor allows the user to choose correct values for rnd_i to make this true.

This use was suggested first at (DI PIETRO et al., 2006) where the authors applied the concept of hash chains to chameleon hash functions. Comparing to regular hash chains, chameleon hash chains have the advantage of providing forward and backward secrecy and does not have a practical limit on the number of authentications. Any collision-resistant chameleon hash is suitable for this application, the uniformity property is not necessary.

4 PREIMAGE CHAMELEON HASH FUNCTIONS

The chameleon hash functions presented in the previous chapter allowed for second-preimage computing using a trapdoor. However, a subset of such chameleon hash functions allows for the more powerful computation of finding preimages using the trapdoor while still having collision-resistance if the trapdoor is not used. There is no standard name for this property or for such stronger version of chameleon hash functions. They were first mentioned in (SHAMIR; TAUMAN, 2001) where this property was called “inversion property”. In (SCHMIDT-SAMOA; TAKAGI, 2005) this property was present as “strong altering”. In (LU et al., 2019c) such chameleon hash functions were called “chameleon hash plus”. Here we call them “preimage chameleon hash functions” and in this chapter we list some of its constructions, properties and applications.

This chapter is organized as follows:

- In Section 4.1 we list the definition and additional properties for preimage chameleon hash functions,
- In Section 4.2 we present different constructions for preimage chameleon hash functions and present security proofs for them.
- In Section 4.3 we list previous applications for this kind of chameleon hash.

4.1 DEFINITION AND PROPERTIES

We can define a preimage chameleon hash scheme as a chameleon hash scheme CH over (M, R, D) where we also have an additional probabilistic algorithm $CH.\textsc{Preimage}$ such that on input tk, msg and dgt, it returns a $rnd \in R$ such that $CH.Hash(ek, msg, rnd) = dgt$. In other words, as in traditional chameleon hash functions the trapdoor allows for second-preimage calculation, in preimage chameleon hash functions, it allows for first-preimage calculation.

When defining a preimage chameleon hash, we do not need to define explicitly the algorithm $CH.\textsc{Collision}$, as using the $CH.\textsc{Preimage}$ algorithm and the evaluation key we can easily define $CH.\textsc{Collision}$:

$$CH.\textsc{Collision}(tk, msg_1, rnd_1, msg_2)$$
$$=$$
$$CH.\textsc{Preimage}(tk, msg_2, CH.\textsc{Hash}(ek, msg_1, rnd_1))$$

We require the same properties of collision resistance and uniformity for preimage chameleon hash functions. However, some applications of this primitive require a stronger version of the uniformity property called “strong uniformity”:

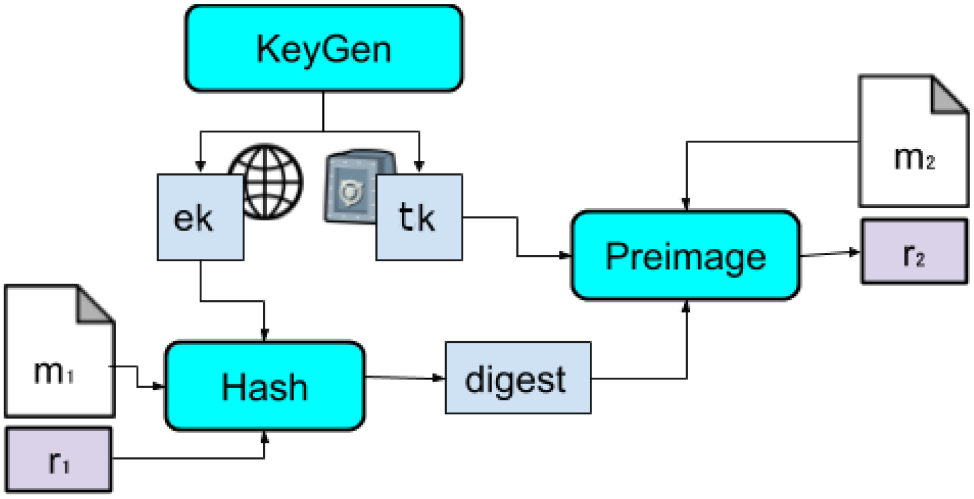

Figure 22 – Diagram representing the algorithms in a preimage chameleon hash scheme. In this image, we have that $Hash(ek, m_1, r_1) = Hash(ek, m_2, r_2)$.

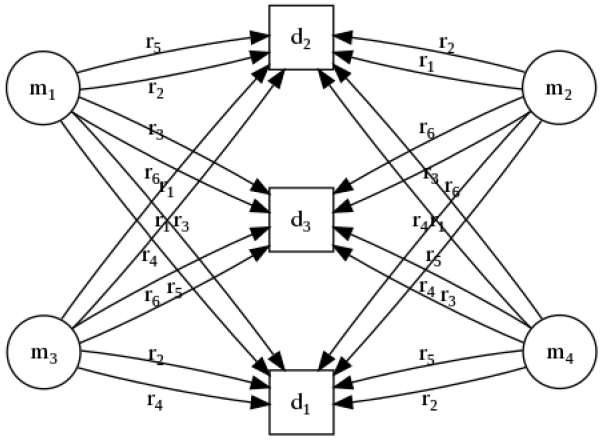

Figure 23 – Representation of a chameleon hash with strong uniformity.

Definition 5 *A preimage chameleon hash scheme Ch defined over* (M, R, D) *has the* ***strong uniformity*** *property if given* $(ek, tk) \xleftarrow{\$} CH.\text{KEYGEN}(\lambda)$*, for all messages* $msg \in M$*, the probability distribution of* $f = CH.\text{HASH}(ek, msg, \cdot)$ *if the input is chosen uniformly at random from R is the same than the probability distribution of choosing an element from D uniformly at random.*

For example, using this definition, both examples of chameleon hash from Figure 19 do not have the strong uniformity property while the following example in Figure 23 shows what is expected from a chameleon hash with strong uniformity: all digests have the same probability of being the hash of any message for a uniformly random $rnd \in R$.

However, contrary to previous chapter, here we will also consider chameleon hash functions with non-uniform distributions if the distribution is negligibly next to a strong uniform distribution.

For a chameleon hash CH over (M, R, D) and any $msg \in M$ and ek returned

by CH.KEYGEN(λ), let $dgt_R \in D$ and $rnd_R \in R$ be elements chosen uniformly at random. Then $Pr[dgt_R = dgt]$ is the probability of picking an specific element $dgt \in D$ when sampling uniformly at random and $Pr[CH.\text{HASH}(ek, msg, rnd_R) = dgt]$ is the probability of picking a specific $dgt \in D$ when computing a chameleon hash with a randomly sampled rnd_R. We compute the statistical distance between the chameleon hash distribution and the strong uniformity distribution as:

$$max_{dgtinD}(|Pr[dgt_R = dgt] - Pr[CH.\text{HASH}(ek, msg, rnd_R) = dgt]|)$$

We consider the chameleon hash distribution negligibly next to strong uniformity if the above statistical distance is negligible (in function of the security parameter λ).

In the above definition, we also do not require that rnd_R is sampled from an uniform distribution. There could be another distribution that results in such property, but we require that this alternative distribution is known for each chameleon hash and also can be efficiently computable.

We also can require in some applications that both CH.HASH and CH.PREIMAGE have random and uniform output if some part of the input is random. We can define the uniformity property for the CH.PREIMAGE algorithm using the definition below.

Definition 6 *We say that a given chameleon hash scheme CH defined over* (M, R, D) *has the* ***preimage uniformity*** *if for all keys tk returned by CH*.KEYGEN *and all messages msg* $\in M$*, if we choose a digest* dgt_R *uniformly and random, then the probability distribution for* $rnd_R \xleftarrow{\$} CH.\text{PREIMAGE}(tk, msg, dgt_R)$ *is indistinguishable from a random and uniform distribution in D.*

If a chameleon hash construction uses a relaxed definition for the uniformity property where the randomness $rnd_R \in R$ is not chosen uniformly at random, but using some distribution $\mathcal{D}$, then for the preimage uniformity we require that given dgt_R chosen randomly, CH.PREIMAGE(tk, msg, dgt_R) produces randomness with a probability distribution indistinguishable from $\mathcal{D}$.

4.2 CONSTRUCTIONS

Here we present three sample constructions of a preimage chameleon hash scheme. One based on the existance of one-way trapdoor function, and two others based on the hardness of the Short Integer Solution (SIS) problem.

4.2.1 Preimage Chameleon Hash from One-Way Trapdoor Permutations

This construction was proposed first as a hash function in (DAMGÅRD, 1987) and was recognized as a chameleon hash at (KRAWCZYK; RABIN, 1998). It uses two one-way trapdoor function schemes OWT_0 and OWT_1 (as defined in Subsection 2.3.3).

Both one-way functions with trapdoor are defined over sets (X, X). We require that $OWT_0.\text{FUNC}(pk, \cdot)$ and $OWT_1.\text{FUNC}(pk, \cdot)$ be **claw-free permutations**, which means that should be hard to find two values $x_1, x_2 \in X$ such that $OWT_1.\text{FUNC}(pk, x_1) = OWT_1.\text{FUNC}(pk, x_2)$.

With the help of these one-way trapdoor functions OWT_1 and OWT_2, we can define a chameleon hash CH_{2OWT} over sets (M, X, X), where $M = \{0,1\}^k$ for some constant k.

The chameleon hash CH_{2OWT} algorithms are defined as:

- $CH_{2OWT}.\text{KEYGEN}(\lambda)$:
  1. $(pk_0, sk_0) \xleftarrow{\$} OWT_0.\text{KEYGEN}(\lambda)$
  2. $(pk_1, sk_1) \xleftarrow{\$} OWT_1.\text{KEYGEN}(\lambda)$
  3. $ek \leftarrow (pk_0, pk_1)$
  4. $tk \leftarrow (sk_0, sk_1)$
  5. **return** (ek, tk)
- $CH_{2OWT}.\text{HASH}(ek, \mathbf{m}, x)$:
  1. $y \leftarrow x$
  2. **for** i **in** $(1, \ldots, k)$:
  3. **if** $\mathbf{m}[i] = 0$:
  4. $y \leftarrow OWT_0.\text{FUNC}(pk_0, y)$
  5. **else if** $\mathbf{m}[i] = 1$:
  6. $y \leftarrow OWT_1.\text{FUNC}(pk_1, y)$
  7. **return** y
- $CH_{2OWT}.\text{PREIMAGE}(tk, \mathbf{m}, y)$:
  1. $x \leftarrow y$
  2. **for** i **in** $(k, \ldots, 1)$:
  3. **if** $\mathbf{m}[i] = 0$:
  4. $x \leftarrow OWT_0.\text{INV}(sk_0, x)$
  5. **else if** $\mathbf{m}[i] = 1$:
  6. $x \leftarrow OWT_1.\text{INV}(sk_1, x)$
  7. **return** x

Theorem 13 *The chameleon hash scheme CH_{2OWT} defined above over sets (M, X, X) has the strong uniformity property and the preimage uniformity.*

Proof: As both $OWT_0.Func$ and $OWT_1.Func$ are permutations, composing these functions as we do to compute CH_{2OWT}.HASH also yields a new permutation. Independent of the message input or the key ek, the result is always a permutation over X having the random parameter $x \in X$ as input.

Therefore, if we choose x uniformly at random, and run CH_{2OWT}.HASH$(ek, \mathbf{m}, x)$, the output of this algorithm will also be random and uniform. This is a sufficient condition to have the strong uniformity property.

The same argument applies for CH_{2OWT}.PREIMAGE, as if OWT_0.FUNC and OWT_1.FUNC are permutations, then so are OWT_0.INV and OWT_1.INV. Therefore, this construction also has preimage uniformity.

■

Theorem 14 *The chameleon hash CH_{2OWT} defined above is collision-resistant if the one-way trapdoor permutations OWT_1 and OWT_2 are claw-free.*

Proof: If we have distint two colliding inputs $(\mathbf{m}_1, x_1)$ and $(\mathbf{m}_2, x_2)$ for algorithm CH_{2OWT}.HASH, then necessarily $\mathbf{m}_1 \neq \mathbf{m}_2$. If the messages were equal, they would represent the same composition of permutation functions, which would also result in a permutation. However, if CH_{2OWT}.HASH$(ek, \mathbf{m}_1, x_1) = CH_{2OWT}$.HASH$(ek, \mathbf{m}_2, x_2)$ for $x_1 \neq x_2$, then this would mean that applying the same permutation to two different inputs result in the same value, which is impossible.

If $\mathbf{m}_1 \neq \mathbf{m}_2$ and they collide in the above hash function, this means that while we are computing their hash, exist an iteration in line 2 of CH_{2OWT}.HASH where OWT_0.FUNC$(x_i) = y^*$ and OWT_1.FUNC$(x_j) = y^*$, where $x_i \neq x_j$. Therefore, this iteration of the CH_{2OWT}.HASH algorithm yields different inputs for OWT_0.FUNC$(pk_0, \cdot)$ and OWT_1.FUNC$(pk_1, \cdot)$ that result in the same value.

By our hypothesis, both trapdoor functions are claw-free and so it is hard to find such input. Therefore, it is also hard finding collisions in CH_{2OWT}. ■

One drawback of this construction is that there are not many known constructions of trapdoor permutations that are also claw-free. One possibility is computing x^2 under modular exponentiation like suggested in the previous subsection. If the modulus is semiprime, it can be proven using the factoring assumption that such exponentiations are claw-free one-way functions.

We will define a suitable one-way trapdoor permutation function OWT_0 and OWT_1, both defined over sets of quadratic residues modulo $n = pq$. We define them as:

- OWT_0.KEYGEN(λ) and OWT_1.KEYGEN are the same and both schemes should share the same keys instead of generating one pair of keys for each of them. They run algorithm GENPRIME(λ) twice and repeat this until finding primes p and

q such that $p \equiv 3 \mod 8$ and $q \equiv 7 \mod 8$. Then they set $n = pq$ The secret key sk is (p, q). The public key pk is n.

- $OWT_0.\text{FUNC}(pk, x) = x^2 \mod n$
- $OWT_1.\text{FUNC}(pk, x) = 4x^2 \mod n$
- $OWT_0.\text{INV}(sk, y)$ and $OWT_1.\text{INV}(sk, y)$ can be computed solving respectively the modular square root of y and $y(4)^{-1}$ modulo n. This can be done solving it in modulo p and in modulo q, combining these results with the Chinese remainder theorem to find the square roots modulo n.

The above restrictions that $p \equiv 3 \mod 8$ and $q \equiv 7 \mod 8$ are necessary to ensure that both $OWT_0.\text{FUNC}$ and $OWT_1.\text{FUNC}$ are permutations. This also ensures that 2 is not a quadratic residue modulo p and –2 is not quadratic residue modulo q, properties that ensure that the construction provides two claw-free permutations as shown in the proof of the theorem below.

Theorem 15 *The above constructions of* $OWT_0.\text{FUNC}$ *and* $OWT_1.\text{FUNC}$ *are claw-free.*

Proof: Let's assume that we found $x_1 \neq x_2$ such that $OWT_0.\text{FUNC}(pk, x_1)$ collide with $OWT_1.\text{FUNC}(pk, x_2)$. This means that:

$$\begin{aligned} x_1^2 &\equiv 4x_2^2 \pmod{pq} \\ x_1^2 - 4x_2^2 &\equiv 0 \pmod{pq} \\ (x_1 + 2x_2)(x_1 - 2x_2) &\equiv 0 \pmod{pq} \end{aligned}$$

This means that if $x_1 \neq 2x_2$ and $x_1 \neq -2x_2$, they are nontrivial factor of a multiple of pq and this formula yields a factorization of n. We know that $x_1 \neq 2x_2$, as 2 is not a quadratic residue modulo p. We know that $x_1 \neq -2x_2$, as -2 is not a quadratic residue modulo q. Therefore, if we could find x_1 and x_2 such that $x_1^2 \equiv 4x_2^2 \pmod{pq}$ with non-negligible probability, then we could find a factorization of pq with non-negligible probability. The proves that these functions are claw-free. ■

The above proof show that if we could find collisions in the presented chameleon hash scheme, we could find a factorization for $n = pq$ if if $p \equiv 3 \mod 8$ and $q \equiv 7 \mod 8$. If we choose primes p and q at random, we expect that these conditions are satisfied with probability 1/16 (as both p and q could be congruent to 1, 3, 5 or 7 in modulo 8 when chosen randomly). Therefore, the upper bound for the probability of factoring a random semiprime n would be $(1/16)CRadv[\mathcal{A}, CH_{2OWT}]$when we use an adversary $\mathcal{A}$ that finds collisions in CH_{2OWT}.

Rewriting this, we can conclude that for all adversaries $\mathcal{A}$ that can find collisions in CH_{2OWT}, we can produce adversaries $\mathcal{B}$ that can factorise semiprimes such that:

$$CRadv[\mathcal{A}, CH_{2OWT}] \leq 16 FACTadv[\mathcal{B}]$$

By the factoring assumption, it is hard to factorise semiprimes and therefore, all adversaries that find collisions in CH_{2OWT} can succeed only with negligible probability. ■

4.2.2 Preimage Chameleon Hash based on SIS Assumption

It is possible to define preimage chameleon hash functions using the SIS assumption (from Section 2.2.3). One advantage is that using these assumptions, we can guarantee the security of the chameleon hash against quantum adversaries, as no quantum or classical algorithm is known to break that assumption.

Before defining the chameleon hash, we will need some additional building blocks presented in (MICCIANCIO; PEIKERT, 2012) that we enumerate and define below.

4.2.2.1 Generating a Random Matrix with Trapdoor: TRAPGEN

We will define here a possible construction for an algorithm that we will denote by TRAPGEN. This algorithm takes as input parameters $(m, n, q, \mathcal{D})$, and outputs a pair of matrices $(\mathbf{A}, \mathbf{T})$ such that $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is indistinguishable from a random and uniform matrix and $\mathbf{T}$ will be a trapdoor that we will use in the next algorithm that was randomly chosen using distribution $\mathcal{D}$. We will need in the construction a constant matrix $\mathbf{G}$. The algorithm works as below:

- TRAPGEN$(m, n, q, \mathcal{D})$:
  1. $\overline{\mathbf{A}} \xleftarrow{\$} \mathbb{Z}_q^{n \times (m - n \lg q)}$
  2. $\mathbf{T} \xleftarrow{\mathcal{D}} \mathbb{Z}_q^{(m - n \lg q) \times (n \lg q)}$
  3. $\mathbf{A} \leftarrow [\overline{\mathbf{A}} | \mathbf{G} - \overline{\mathbf{A}}\mathbf{T}]$
  4. **return** $(\mathbf{A}, \mathbf{T})$

Notice that in the above algorithm , matrix $\mathbf{G}$ is a $n \times (n \lg q)$ matrix and the generated matrix $\mathbf{A}$ is $n \times m$ as required..

It is believed that the matrix $\mathbf{A}$ as computed above is indistinguishable from a random matrix for appropriate input parameters. The first m' columns indeed were chosen uniformly at random. The remaining columns can be considered as the output of a possible pseudo-random function *PRF* defined below:

$$PRF_{\text{TRAPGEN}}(\mathbf{T}, \overline{\mathbf{A}}) = \mathbf{G} - \overline{\mathbf{A}}\mathbf{T}$$

In fact, this *PRF* can be a weak *PRF*: it should be indistinguishable from a random function provided that its input are always random and uniform, not necessarily for any chosen input.

4.2.2.2 Finding Short Integer Solutions: SAMPLEPRE

Now we are interested in solving the Short Integer Solution Problem for matrices $\mathbf{A}$ returned by the TRAPGEN algorithm. Given the matrix and the trapdoor, we want to find short vectors $\mathbf{x}$ such that $\mathbf{Ax} = 0 \pmod{q}$. More than this, we want to be able to find short vectors $\mathbf{x}_i$ such that for any arbitrary $\mathbf{y}_i \in \mathbb{Z}_q^n$ we have $\mathbf{Ax}_i = \mathbf{y}_i \pmod{q}$.

We will define a possible implementation for an algorithm SAMPLEPRE that given $(m, n, q, \mathbf{A}, \mathbf{T}, \mathbf{y})$ produce a $\mathbf{x}$ such that $\mathbf{Ax} = \mathbf{y} \pmod{q}$ and $\mathbf{x}$ has a short euclidean norm.

First notice that for any $\mathbf{A} = \left[\overline{\mathbf{A}} \quad \mathbf{G} - \overline{\mathbf{A}}\mathbf{T}\right]$ returned by TRAPGEN algorithm, we have that:

$$\left[\overline{\mathbf{A}} \quad \mathbf{G} - \overline{\mathbf{A}}\mathbf{T}\right] \begin{bmatrix}\mathbf{T}\\ \mathbf{I}\end{bmatrix} = \overline{\mathbf{A}}\mathbf{T} + \mathbf{G} - \overline{\mathbf{A}}\mathbf{T} = \mathbf{G}$$

We defined the matrix $\mathbf{G}$ as any arbitrary non-random constant. We can assume that we can choose a suitable matrix $\mathbf{G}$ such that for this matrix it is easy to find short vectors $\mathbf{x}$ such that $\mathbf{Gx}$ is any desired result.

Using these properties, we can provide a possible implementation for SAMPLEPRE:

- SAMPLEPRE$(m, n, q, \mathbf{A}, \mathbf{T}, \mathbf{y})$:
  1. $\mathbf{p} \xleftarrow{\mathcal{D}} \mathbb{Z}_q^m$
  2. $\mathbf{v} \leftarrow \mathbf{y} - \mathbf{Ap} \mod q$
  3. Find a small $\mathbf{z}$ such that $\mathbf{Gz} = \mathbf{v} \mod q$
  4. $\mathbf{x} \leftarrow \mathbf{p} + \begin{bmatrix}\mathbf{T}\\ \mathbf{I}\end{bmatrix} \mathbf{z}$
  5. If $||\mathbf{x}||$ is not small enough, return to step 1
  6. **return x**

In this algorithm, in line 1 we choose a small perturbation randomly using some Gaussian distribution that return small values with higher probability. The size of the obtained $\mathbf{x}$ depends on how short is the perturbation, the vector $\mathbf{z}$ produced in line 3 and also depends on the quality of the trapdoor $\mathbf{T}$ produced by TRAPGEN.

To prove that the algorithm works, notice that multiplying $\mathbf{A}$ by a vector returned by SAMPLEPRE$(m, n, q, \mathbf{A}, \mathbf{T}, \mathbf{y})$ we really obtain $\mathbf{y}$:

$$\begin{aligned}
\mathbf{Ax} \mod q &= \mathbf{A}(\mathbf{p} + \begin{bmatrix}\mathbf{T}\\ \mathbf{I}\end{bmatrix}\mathbf{z}) \\
&= \mathbf{Ap} + \mathbf{A}\begin{bmatrix}\mathbf{T}\\ \mathbf{I}\end{bmatrix}\mathbf{z} \\
&= \mathbf{Ap} + \mathbf{Gz} \\
&= \mathbf{Ap} + \mathbf{v} \\
&= \mathbf{Ap} + \mathbf{y} - \mathbf{Ap} \\
&= \mathbf{y}
\end{aligned}$$

When we use SAMPLEPRE in a construction, usually we have a maximum value for the euclidean norm of vectors returned by SAMPLEPRE. We will make these maximum norms explicit as subscripts when we invoke this algorithm. For example, if we write $\mathbf{x} \xleftarrow{\$} \text{SAMPLEPRE}_{\beta/2}(m, n, q, \mathbf{A}, \mathbf{T}, \mathbf{y})$, this means that we are using some implementation that will always return values with norm smaller than $\beta/2$.

Finally, we need to show how we could choose a matrix $\mathbf{G}$ such that for any $\mathbf{y}$ we can easily find short vectors $\mathbf{x}$ that $\mathbf{Gx} = \mathbf{y} \pmod q$. By the previous algorithms, $\mathbf{G}$ must be a $n \times n \lg q$ matrix.

Assume that q is a power of 2. And for example, we have $q = 32$ and $n = 3$. In this case, we could build the following matrix:

$$G = \begin{bmatrix}
1 & 2 & 4 & 8 & 16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 8 & 16 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 8 & 16
\end{bmatrix}$$

Notice that given any $\mathbf{y}$, we can find a suitable binary vector $\mathbf{x}$ such that $\mathbf{Gx} = \mathbf{y}$ letting $\mathbf{x}$ be a binary representation for each element in $\mathbf{y}$. For example, if $\mathbf{y} = \begin{bmatrix}2 & 5 & 10\end{bmatrix}^T$, then we have:

$$\mathbf{G}\begin{bmatrix}0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0\end{bmatrix}^T = \begin{bmatrix}2\\5\\10\end{bmatrix}$$

4.2.2.3 Sampling Short Random Vectors: SAMPLEDOM

In our next constructions we will need to sample short vectors $\mathbf{x}$ such that the result $\mathbf{Ax}$ is indistinguishable from a random and uniform vector $\mathbf{y} \in \mathbb{Z}_q^n$. For this property, we cannot choose $\mathbf{x}$ uniformly at random from the set of sufficiently short vectors.

To get suitable vectors $\mathbf{x}$ with such property, we will assume that we have two possible methods. First algorithm $\text{SAMPLEPRE}(m, n, q, \mathbf{A}, \mathbf{T}, \mathbf{y})$ return a possible $\mathbf{x}$ with such property if we first choose $\mathbf{y}$ uniformly at random. The problem with this method is that it requires the trapdoor $\mathbf{T}$.

For the second method, we will assume that we have an algorithm invoked as $\mathbf{x} \xleftarrow{\$} \text{SAMPLEDOM}(m, n, q, \mathbf{A})$ that return a suitable short vector with such property. In practice this algorithm is implemented sampling a short vector using a discrete Gaussian distribution.

We also assume that SAMPLEDOM produce as output vectors $\mathbf{x} \in \mathbb{Z}_q^m$ with a probability distribution indistinguishable from what would be obtained choosing a random and uniform $\mathbf{y} \in \mathbb{Z}_q^m$ and computing $\text{SAMPLEPRE}(m, n, q, \mathbf{A}, \mathbf{T}, \mathbf{y})$.

Like the SAMPLEPRE algorithm, if we are invoking a specific implementation of SAMPLEPRE where we expect results with norm smaller than some value, we make this explicit with a subscript. For example, in $\text{SAMPLEDOM}_\beta(m, n, q, \mathbf{A})$ we are referring to a implementation that always return vectors with norm smaller than β.

4.2.2.4 The Chameleon Hash Construction

Using the previous algorithms TRAPGEN, SAMPLEPRE and SAMPLEDOM, we can build a chameleon hash denoted by CH_{SIS} over sets (M, R, D).

Here $M = \{0,1\}^k$ is a binary vector for some constant k, R is the set of vectors from $\mathbb{Z}^{m'+n\lg q}$ with euclidean norm smaller than $1/2\sqrt{\beta^2 - k}$ and $D = \mathbb{Z}_q^n$. The chameleon hash algorithms are:

- CH_{SIS}.KEYGEN(λ):
  1. $(m, n, q, \beta) \leftarrow \text{SISPARAMS}(\lambda)$
  2. $\mathbf{A}_1 \xleftarrow{\$} \mathbb{Z}_q^{n \times k}$
  3. $(\mathbf{A}_2, \mathbf{T}) \xleftarrow{\$} \text{TRAPGEN}(m-k, n, q, \mathcal{D})$
  4. $ek \leftarrow (m, n, q, \beta, \mathbf{A}_1, \mathbf{A}_2)$
  5. $tk \leftarrow (m, n, q, \beta, \mathbf{A}_1, \mathbf{A}_2, \mathbf{T})$
  6. **return** (ek, tk)
- CH_{SIS}.HASH($ek, \mathbf{m}, \mathbf{r}$):
  1. $(m, n, q, \beta, \mathbf{A}_1, \mathbf{A}_2) \leftarrow ek$
  2. **return** $\mathbf{A}_1\mathbf{m} + \mathbf{A}_2\mathbf{r} \mod q$
- CH_{SIS}.PREIMAGE($tk, \mathbf{d}, \mathbf{m}$):
  1. $(m, n, q, \beta, \mathbf{A}_1, \mathbf{A}_2, \mathbf{T}) \leftarrow tk$
  2. **return** $\text{SAMPLEPRE}_{1/2\sqrt{\beta^2-k}}(m-k, n, q, \mathbf{A}_2, \mathbf{T}, \mathbf{d} - \mathbf{A}_1\mathbf{m})$

This construction works because if $\mathbf{r} = CH_{SIS}.\text{PREIMAGE}(ek, \mathbf{m}, \mathbf{d})$, then:

$$\begin{aligned} CH_{SIS}.\text{HASH}(ek, \mathbf{m}, \mathbf{r}) &= \mathbf{A}_1\mathbf{m} + \mathbf{A}_2\mathbf{r} \mod q \\ &= \mathbf{A}_1\mathbf{m} + (\mathbf{d} - \mathbf{A}_1\mathbf{m}) \mod q \\ &= \mathbf{d} \end{aligned}$$

However, to ensure the strong uniformity property, we cannot simply choose a short vector $\mathbf{r}$ from a uniform distribution. We should always use the algorithm $\text{SAMPLEDOM}_{1/2\sqrt{\beta^2-k}}(m, n, q, \mathbf{A})$.

Theorem 16 *If the chameleon hash CH_{SIS} always has the randomness $\boldsymbol{r}$ chosen using $\text{SAMPLEDOM}_{1/2\sqrt{\beta^2-k}}(m, n, q, \boldsymbol{A})$, then it has the strong uniformity property and the preimage uniformity.*

Proof: When computing the CH_{SIS}.HASH algorithm, if the randomness was chosen by algorithm SAMPLEDOM, by definition, $\mathbf{A}_2\mathbf{r}$ is a random and uniform vector in $\mathbb{Z}_n$. Therefore, when we sum $\mathbf{Am}$ with this value, the resulting digest remains as a random and uniform value in $\mathbb{Z}_n$.

For the preimage uniformity, this is a consequence of how we specified algorithm SAMPLEDOM: It generate values using exactly the same probability distribution than computing SAMPLEPRE with a random and unioform $\mathbf{y}$. Therefore, in the definition of CH_{SIS}.PREIMAGE algorithm, in line 2, id $\mathbf{d}$ is a random and uniform value, then $\mathbf{d} - \mathbf{A}_1\mathbf{m}$ is also random and uniform. Therefore, given a random and uniform digest, CH_{SIS}.PREIMAGE would compute using SAMPLEPRE a randomness with the same distribution than SAMPLEDOM.

■

The collision-resistance of this construction can be proven using the SIS assumption.

Theorem 17 *The above chameleon hash CH_{SIS} is collision-resistant if the SIS assumption is true and if the algorithm* TRAPGEN *generates a matrix indistinguishable from a random matrix.*

Proof: First we will assume that our adversary that finds collisions for the chameleon hash CH_{SIS} cannot distinguish between $\mathbf{A}_2$ and a random and uniform matrix. We will denote by $\mathcal{A}_1$ any adversary with such characteristics.

Using an adversary $\mathcal{A}_1$, we will build an adversary $\mathcal{B}$ that uses $\mathcal{A}_1$ to solve the SIS assumption. Adversary $\mathcal{B}$ acts in the following way:

- **Initialization**: Adversary $\mathcal{B}_1$ takes as input from its challenger a tuple $(m, n, q, \beta, \mathbf{A})$. It separates matrix A in two matrices $(\mathbf{A}_1, \mathbf{A}_2)$. The first k columns became matrix

$\mathbf{A}_1$ and the remaining columns became matrix $\mathbf{A}_2$.Finally, it initializes adversary $\mathcal{A}_\infty$ with the key $ek = (m, n, q, \beta, \mathbf{A}_1, \mathbf{A}_2)$.

- **Finalization:** After adversary $\mathcal{A}_1$ outputs a possible collision (**m'**, **r'**) and (**m''**, **r''**), adversary $\mathcal{B}$ sends (**m'**||**r'**) – (**m''**||**r''**) as a solution for the SIS attack game.

The idea is that if $\mathbf{A}_1\mathbf{m'} + \mathbf{A}_2\mathbf{r'} = \mathbf{A}_1\mathbf{m''} + \mathbf{A}_2\mathbf{r''}$ in modulo q, this means that $(\mathbf{A}_1||\mathbf{A}_2)(\mathbf{m'}||\mathbf{r'}) = (\mathbf{A}_1||\mathbf{A}_2)(\mathbf{m''}||\mathbf{r''})$. And therefore: $\mathbf{A}(\mathbf{m'}||\mathbf{r'}) = \mathbf{A}(\mathbf{m''}||\mathbf{r''})$. This means that:

$$\mathbf{A}(\mathbf{m'}||\mathbf{r'}) - \mathbf{A}(\mathbf{m''}||\mathbf{r''}) = 0 \pmod q$$

And therefore, (**m'**||**r'**) – (**m''**||**r''**) is a possible solution for the *SIS* problem. We still need to show that the euclidean norm of this resulting vector is indeed less than β as required by the SIS attack game.

Notice that $||\mathbf{m'} - \mathbf{m''}|| \leq \sqrt{k}$ because they are binary vectors with k bits. In the worst case, **m'** is a vector composed only by 1 and **m''** is a vector composed only by 0, resulting in the euclidean norm $\sqrt{k}$.

We know that both $||\mathbf{r'}||$ and $||\mathbf{r''}||$ have a euclidean norm smaller than $1/2\sqrt{\beta^2 - k}$. By triangle inequality, we can conclude that $||\mathbf{r}_1 - \mathbf{r}_2|| \leq \sqrt{\beta^2 - k}$. Therefore, the norm of the vector returned by $\mathcal{B}_1$ is given by:

$$\sqrt{||\mathbf{m'} - \mathbf{m''}||^2 + ||\mathbf{r'} - \mathbf{r''}||^2} \leq \sqrt{k + \beta^2 - k} \leq \beta$$

This means that adversary $\mathcal{B}$ always succeeds if adversary $\mathcal{A}_1$ succeeds. However, this proof is only for adversaries that try to find collisions for CH_{SIS} without trying to explore any possible difference between matrix $\mathbf{A}_2$ and a random and uniform matrix. A more general adversary $\mathcal{A}$ could also try to find a weakness in PRF_{TRAPGEN} defined in Subsection 4.2.2.1 and explore some pattern not present in a random and uniform matrix. The probability of finding such difference between $\mathbf{A}_2$ and a random and uniform matrix is given by $WPRFadv[\mathcal{B}', PRF_{\text{TRAPGEN}}]$ for some adversary $\mathcal{B}'$.

Therefore, to model the upper bound for the probability of success for the above adversary $\mathcal{B}$ that solves the SIS assumption, we need to discard the probability that $\mathcal{A}$ finds a collision exploring a weakness in PRF_{TRAPGEN}, as such weakness will not be found in matrices sent by $\mathcal{B}$:

$$SISadv[\mathcal{B}] \geq CRadv[\mathcal{A}, CH_{SIS}] - WPRFadv[\mathcal{B}', PRF_{\text{TRAPGEN}}]$$

Therefore, for all efficient adversaries $\mathcal{A}$ that find collisions in CH_{SIS}, we can build efficient adversaries $\mathcal{B}$ and $\mathcal{B}'$ such that:

$$CRadv[\mathcal{A}, CH_{SIS}] \leq SISadv[\mathcal{B}] + PRFadv[\mathcal{B}', PRF_{\text{TRAPGEN}}]$$

By our assumption, the Short Integer Solution (SIS) assumption is hard, therefore $SISadv[\mathcal{B}]$ is negligible. And also we assume that the algorithm TRAPGEN generates

a matrix computationally indistinguishable from a random and uniform matrix, which means that $WPRFadv[\mathcal{B}', PRF_{\text{TRAPGEN}}]$ is also negligible. Therefore, no adversary can find collisions in CH_{SIS}, except with negligible probability. ■

4.2.3 A Second Chameleon Hash Based on SIS Assumption

This construction was first proposed at (GORBUNOV et al., 2015). The author presented the construction not as a chameleon hash, but as a new proposed primitive called "homomorphic trapdoor functions" (HTDF). However, that construction fits in our definition of chameleon hash.

To build this scheme, we will generalize a little the previously defined algorithms SAMPLEPRE and SAMPLEDOM producing a matrix version of these algorithms. To differentiate between the versions, we will call the matrix versions MATRIXSAMPLEPRE and MATRIXSAMPLEDOM.

Instead of finding a short vector $\mathbf{x}$ such that $\mathbf{Ax} = \mathbf{y}$, algorithm MATRIXSAMPLEPRE finds a matrix $\mathbf{X}$ such that $\mathbf{AX} = \mathbf{Y}$. And algorithm MATRIXSAMPLEDOM returns a matrix $\mathbf{X}$ such that $\mathbf{AX}$ will have a random and uniform distribution. Here $\mathbf{X}, \mathbf{Y} \in \mathbb{Z}_q^{m \times m}$. The new algorithms can easily be implemented iterating m times original versions SAMPLEPRE and SAMPLEDOM.

Finally, we will not use the euclidean norm to measure the size of our vectors and matrices. Our norm will be equal the greatest value in a vector or matrix. This will allow us to define more easily the norm of matrices. Recall that we denote this norm by $||\mathbf{M}||_\infty$.

Using these matrix version of the algorithms, we will denote the new chameleon hash construction by CH_{HTDF}. The chameleon hash is defined over (M, R, D) where $M = \mathbb{Z}_q$, R is the set of square matrices from $\mathbb{Z}_q^{m \times m}$ whose columns are vectors with norm smaller than $(\sqrt{8m\beta + 1} - 1)/4m$ and $D = \mathbb{Z}_q^{n \times m}$. We will also use a matrix $\mathbf{G}$ such that it is easy to find short vectors $\mathbf{x}$ such that $\mathbf{Gx}$ is any desired result.

The algorithms of this construction are defined as below:

1. CH_{HTDF}.KEYGEN(λ):
   a) $(m, n, q, \beta) \xleftarrow{\$} SISParams(\lambda)$
   b) $(\mathbf{A}, \mathbf{T}) \xleftarrow{\$} \text{TRAPGEN}(m, n, q, \beta, \mathcal{D})$
   c) $ek \leftarrow (m, n, q, \beta, \mathbf{A})$
   d) $tk \leftarrow (m, n, q, \beta, \mathbf{A}, \mathbf{T})$
   e) **return** (ek, tk)
2. CH_{HTDF}.HASH($ek, x, \mathbf{R}$):
   a) $(m, n, q, \beta, \mathbf{A}) \leftarrow ek$

b) **return** $x\mathbf{G} + \mathbf{AR} \mod q$

3. CH_{HTDF}.PREIMAGE$(tk, x, \mathbf{D})$:

   a) $(m, n, q, \beta, \mathbf{A}, \mathbf{T}) \leftarrow tk$

   b) **return** MATRIXSAMPLEPRE$_{(\sqrt{8m\beta+1}-1)/4m}(m, n, q, \mathbf{A}, \mathbf{T}, \mathbf{D} - x\mathbf{G})$

This construction works because if $\mathbf{R} \xleftarrow{\$} CH_{HTDF}$.PREIMAGE$(tk, x, \mathbf{D})$, then:

$$\begin{aligned} CH_{HTDF}.\text{HASH}(ek, x, \mathbf{R}) &= x\mathbf{G} + \mathbf{AR} \\ &= x\mathbf{G} + \mathbf{D} - x\mathbf{G} \\ &= \mathbf{D} \end{aligned}$$

Like in the previous construction, we guarantee the strong uniformity property and the preimage uniformity only if the randomness used to compute hash functions is chosen by algorithm MATRIXSAMPLEDOM. The proof for the uniformity property is analogous to the proof for CH_{SIS}.

The proof for the collision-resistance is given below.

Theorem 18 *If the SIS assumption is true and if the matrix* $\boldsymbol{A}$ *returned by* TRAPGEN *is indistinguishable from a random matrix, then the chameleon hash* CH_{HTDF} *is collision-resistant.*

Proof: If we have an algorithm $\mathcal{A}$ that outputs a collision for CH_{HTDF} in the collision-resistance attack game, it is possible to use it to build an adversary $\mathcal{B}$ that wins the SIS attack game.

If $\mathcal{A}$ return a real collision $(x', \mathbf{R'})$ and $(x'', \mathbf{R''})$, then:

$$\begin{aligned} x'\mathbf{G} + \mathbf{AR'} &= x''\mathbf{G} + \mathbf{AR''} \mod q \\ (x' - x'')\mathbf{G} &= \mathbf{A}(\mathbf{R'} - \mathbf{R''}) \mod q \end{aligned}$$

If $m' = m'$, then $\mathbf{A}(\mathbf{R'} - \mathbf{R''}) \mod q = 0$. Then we can find a solution for the SIS attack game picking any column from matrix $(\mathbf{R'} - \mathbf{R''})$.

If $m' \neq m''$, we can pick some vector $\mathbf{r} \in \mathbb{Z}_q^m$ with norm smaller than $\beta/(3m+1)$ and another vector $\mathbf{r'} \in \mathbb{Z}_q^m$ such that $G\mathbf{r'} = A\mathbf{r} \mod q$. Now we can use the previous inequation, multiplying both sides by $\mathbf{r}$ and subtracting both sides by $A\mathbf{r'}$ to obtain:

$$
\begin{aligned}
(x' - x'')\mathbf{G} &= \mathbf{A}(\mathbf{R'} - \mathbf{R''}) \mod q \\
(x' - x'')\mathbf{Gr'} &= \mathbf{A}(\mathbf{R'} - \mathbf{R''})\mathbf{r'} \mod q \\
(x' - x'')\mathbf{Ar} &= \mathbf{A}(\mathbf{R'} - \mathbf{R''})\mathbf{r'} \mod q \\
\mathbf{Ar} &= (x' - x'')^{-1}\mathbf{A}(\mathbf{R'} - \mathbf{R''})\mathbf{r'} \mod q \\
0 &= (x' - x'')^{-1}\mathbf{A}(\mathbf{R'} - \mathbf{R''})\mathbf{r'} - \mathbf{Ar} \mod q \\
0 &= (x' - x'')^{-1}\mathbf{A}((\mathbf{R'} - \mathbf{R''})\mathbf{r'} - \mathbf{r}) \mod q
\end{aligned}
$$

Which means that a solution for the SIS attack game can be obtaining computing $(\mathbf{R'} - \mathbf{R''})\mathbf{r'} - \mathbf{r}$.

About the norm of the returned value, we know that $||\mathbf{R'}||_\infty < (\sqrt{8m\beta + 1} - 1)/4m$ and $||\mathbf{R''}||_\infty < (\sqrt{8m\beta + 1} - 1)/4m$. Therefore, $||\mathbf{R'} - \mathbf{R''}|| < (\sqrt{8m\beta + 1} - 1)/2m$. To find the upper bound for the norm $||(\mathbf{R'} - \mathbf{R''})\mathbf{r'}||_\infty$, recall that in the worst case a matrix and the vector has all its elements equal its norm. So for any matrix $\mathbf{M}$ and vector $\mathbf{x}$ we have the upper bound $||\mathbf{Mx}||_\infty \leq m||\mathbf{M}||_\infty||\mathbf{x}||_\infty$ where m is the number of elements in $\mathbf{x}$. This means that:

$$||(\mathbf{R'} - \mathbf{R''})\mathbf{r'}||_\infty \leq m||(\mathbf{R'} - \mathbf{R''})||_\infty||\mathbf{r'}||_\infty = \frac{8m\beta + 2 - 2\sqrt{8m\beta + 1}}{8m}$$

Finally, if we subtract $\mathbf{r'}$, which also has norm smaller than $(\sqrt{8m\beta + 1} - 1)/4m$ we can compute the norm of the solution as:

$$||(\mathbf{R'} - \mathbf{R''})\mathbf{r'} - \mathbf{r}||_\infty \leq \frac{8m\beta + 2 - 2\sqrt{8m\beta + 1}}{8m} + \frac{\sqrt{8m\beta + 1} - 1}{4m} = \beta$$

However, there is a small probability that $(\mathbf{R'} - \mathbf{R''})\mathbf{r'} - \mathbf{r} = 0$, and in this case, we fail to produce a solution for the SIS attack game. This probability was shown in (GORBUNOV et al., 2015) to be $O(2^{-\log\lambda})$. In this unlikely scenario, we could keep generating new values for $\mathbf{r}$ and compute corresponding $\mathbf{r'}$ until finding a suitable solution. We will represent the negligible probability of not finding a solution given a valid collision after a polynomially bounded number of tries by the negligible value ϵ.

Therefore, if we have an adversary $\mathcal{A}$ that finds collisions in this chameleon hash, we can build an adversary $\mathcal{B}$ that solves the SIS_∞ assumption such that:

$$SIS_\infty adv[\mathcal{B}] \geq CRadv[\mathcal{A}, CH_{HTDF}] - \epsilon$$

Therefore, for all adversaries $\mathcal{A}$, we can build adversaries $\mathcal{B}$ such that:

$$CRadv[\mathcal{A}, CH_{HTDF}] \leq SIS_\infty adv[\mathcal{B}] + \epsilon$$

By the SIS assumption using the higher element as norm, the value of $SIS_\infty adv[\mathcal{B}]$ is negligible and ϵ is also negligible. Therefore, the probability of any adversary finding a collision in CH_{HTDF} is also negligible. ■

4.2.4 Other Constructions

In (ATENIESE; MEDEIROS, 2004), a preimage chameleon hash construction based on discrete logarithm was proposed. That construction had a security proof that the chameleon hash remained secure even if sample collisions are leaked to attackers. Even after observing a polynomial number of collisions, adversaries could not find new collisions, except with negligible probability. The security proof was in the generic group model, a non-standard model like the random oracle model. However, the proof is not valid against adversaries with access to quantum algorithms.

In (SCHMIDT-SAMOA; TAKAGI, 2005), an alternative construction of chameleon hash functions based on the difficulty of factoring p^2q was also proposed. The alternative was presented as better suited for using on on-line/off-line signatures.

4.3 APPLICATIONS

Most proposed applications for chameleon hash functions do not need the stronger property of computing preimages. This explains why despite a lot of existing chameleon hash functions having this property, usually they are presented as regular chameleon hash functions.

The exception is the application described below of how to use preimage chameleon hashes to increase the security of an existing signature scheme, even more than what is possible with regular chameleon hashes, There are also some applications for preimage chameleon hash functions in the building of signature schemes. We let these applications to the next chapter where they can be presented in more detail.

4.3.1 Transforming RMA-Secure Signatures in CMA-Secure Signatures

In Subsection 3.4.3 we presented how using a chameleon hash one could transform signatures secure against generic chosen message attacks in signatures secure against adaptive chosen message attacks. However, using preimage chameleon hash functions an even greater increase in security can be achieved. Using the same transformation we can transform a signature secure against random message attacks in a signature secure against adaptive chosen message attacks. The technique described below was first presented at (SHAMIR; TAUMAN, 2001).

Let SIG be a signature scheme defined over (D, S) secure against random message attacks (as described in Section 2.3.5). Let CH be a collision-resistant preimage chameleon hash defined over (M, R, D) with strong uniformity property. Let's define a new signature scheme denoted by $SIG \circ CH$ exactly as we defined in Subsection 3.4.3, but using a preimage chameleon hash instead of a regular chameleon hash.

Theorem 19 *If SIG is a signature scheme secure against random message attacks and*

CH is a collision-resistant chameleon hash with preimage uniformity, then the signature scheme (*SIG* ∘ *CH*) *defined above is secure against adaptive chosen message attack.*

Proof: We will show how using an adversary $\mathcal{A}$, which can forge a signature for (*SIG* ∘ *CH*) in the adaptive chosen message attack game, we could build an adversary $\mathcal{B}$ that forges a signature against *SIG* in the random message attack game.

Our forger $\mathcal{B}$ is composed of the following parts:

- **Initialization:** Here $\mathcal{B}$ is initialized with the security parameter λ and takes as input from its challenger the public key *pk* for *SIG*. It proceeds to run:

  $$(ek, tk) \xleftarrow{\$} CH.KeyGen(\lambda)$$

  . Then it gets from its challenger a polynomial number of tuples of messages and its signatures, where the messages were chosen uniformly at random: $((dgt_1, sig_1), \ldots, (dgt_q, sig_q))$.

  The adversary $\mathcal{B}$ can then produce a valid public key $pk' = (ek, pk)$ for (*SIG* ∘ *CH*). It initializes $\mathcal{A}$ with the security parameter λ and sends pk' to $\mathcal{A}$.

- **Queries:** The adversary $\mathcal{A}$ will send a polynomial number of queries. In each of them, it will send a message msg_i. The adversary $\mathcal{B}$ produces a response computing $rnd_i \leftarrow CH.\textsc{Preimage}(tk, msg_i, dgt_i)$ and sending the signature (sig_i, rnd_i) to $\mathcal{A}$. Notice that here the preimage uniformity is required, or $\mathcal{B}$ would not simulate correctly a legitimate signer, as the produced rnd_i would have a probability distribution different than expected.

- **Finalization:** After all the queries, the adversary $\mathcal{A}$ produces an attempted forgery $(msg', (sig', rnd'))$. First the adversary $\mathcal{B}$ checks if the forgery was successful. If not, it halts. If yes, it produces a forgery for *SIG* sending $(CH.\textsc{Hash}(ek, msg', rnd'), sig')$ to its challenger.

The above adversary $\mathcal{B}$ fails if $\mathcal{A}$ fails or if the digest $CH.\textsc{Hash}(ek, msg', rnd')$ was present in the queries sent by the challenger. If this happens, then it means that $\mathcal{A}$ found a forgery in (*SIG* ∘ *CH*) computing a collision in *CH*.

Therefore, the upper bound for the probability of $\mathcal{B}$ winning the attack game is given by the probability of $\mathcal{A}$ finding a forgery minus the probability of this forgery revealing a collision in *CH*, which can be modeled as $CRadv[\mathcal{B}', CH]$ for some $\mathcal{B}'$:

$$RMAadv[\mathcal{B}, SIG] \geq CMAadv[\mathcal{A}, (SIG \circ CH)] - CRadv[\mathcal{B}', CH]$$

Rewriting this inequation, we can conclude that for all adversaries $\mathcal{A}$ that can create forgeries for (*SIG* ∘ *CH*) using an adaptive chosen message attack, we can define adversaries $\mathcal{B}$ and $\mathcal{B}'$ such that:

$$CMAadv[\mathcal{A}, (SIG \circ CH)] \leq RMAadv[\mathcal{B}, SIG] + CRadv[\mathcal{B}', CH]$$

By our assumptions, *SIG* is secure against random message attacks and *CH* is collision-resistant. Therefore, no adversary can create forgeries in $(SIG \circ CH)$ using adaptive chosen message attacks, except with negligible probability. ■

5 SIGNATURES AND CHAMELEON HASH FUNCTIONS

Relationship between chameleon hash functions and signature schemes were observed by some authors. The oldest article found that noticed this relationship was (ZHANG, R., 2007), where the author notes that computing a collision in the chameleon hash could be used as a way to define one-time signatures and that the main difficulty were proving that an adversary would not be able to forge new signatures computing new collisions after having access to a sample collision. Despite making this observation, that article do not develop the concept and suggested this as an open problem.

This chapter deals with other more recent articles that attacked this problem proposed secure schemes of how to use chameleon hash functions to directly build digital signature schemes. In the end we propose a new chameleon hash variant and a novel signature scheme with an interesting property using it.

This chapter is organized as follows:

- Section 5.1 presents the simplest one-time weak signature scheme based in a chameleon hash scheme.
- Section 5.2 presents another one-time signature, but secure in a stronger security model.
- Section 5.3 presents a signature scheme secure against adaptive chosen message attacks assuming that the adversary is classical.
- Section 5.4 presents a universal designated verifier signature. An alternative signature scheme that tries to preserve the signer privacy permitting only a designated person to verify a signed message, similarly to chameleon signatures.
- Section 5.5 presents an homomorphic signature scheme.
- Section 5.6 presents our novel signature scheme, secure against adaptive chosen message attacks in classical and post-quantum model assuming that the Ring-SIS assumption is true.

5.1 ONE-TIME SIGNATURE SECURE AGAINST WEAK CHOSEN-MESSAGE ATTACK

This signature scheme was first presented at (MOHASSEL, 2010). The idea is that if we have a chameleon hash CH defined over (M, R, D), then we can produce a signature for some message $msg \in M$ providing a $rnd \in R$ such that $CH.\text{HASH}(ek, msg, rnd)$ is equal to some constant value predefined.

This could be achieved with the following algorithms that define a signature scheme denoted by SIG_1 defined over (M, R):

- SIG_1.KEYGEN(λ):
  1. $(ek, tk) \xleftarrow{\$} CH$.KEYGEN($\lambda$)
  2. $msg' \xleftarrow{\$} M$
  3. $rnd' \xleftarrow{\$} R$
  4. $dgt \leftarrow CH$.HASH(ek, msg, rnd)
  5. $pk \leftarrow (ek, dgt)$
  6. $sk \leftarrow (tk, msg', rnd')$
  7. **return** (pk, sk)
- SIG_1.SIGN(sk, msg):
  1. $(tk, msg, rnd) \leftarrow sk$
  2. $rnd' \xleftarrow{\$} CH$.COLLISION($tk, msg', rnd', msg$)
  3. $sig \leftarrow rnd'$
  4. **return** sig
- SIG_1.VERIFY(pk, msg, sig):
  1. $(ek, dgt) \leftarrow pk$
  2. **if** CH.HASH(ek, msg, sig) = dgt:
     a) **return** `accept`
  3. **else**:
     a) **return** `reject`

This is a very simple signature scheme, but there are some problems with it. Each new signature after the first could reveal a collision in the chameleon hash. When this happens, we cannot guarantee that some adversary cannot use this information to produce new collisions and forgeries. Because of this, the signature scheme SIG_1 can be used only as a one-time signature, at least for generic constructions that use CH as a black box.

Even as a one-time signature, we can prove its security only in the weaker security model where the adversary must make its signing query before knowing the public key. This is what we called a signature with a weak security in Subsection 2.3.5.1.

Theorem 20 *If the chameleon hash scheme CH defined over* (M, R, D) *is collision-resistant and has the uniformity property, then the signature* SIG_1 *is a one-time signature secure against weak chosen message attacks.*

Proof: We will show how to create an efficient adversary $\mathcal{B}$ that finds a collision in *CH* if we have an efficient adversary $\mathcal{A}$ that forges a signature in SIG_1 using the Attack Game 12 in the weak variant.

Our adversary $\mathcal{B}$ acts in the following way:

- **Initialization:** Adversary $\mathcal{B}$ is initialized by the security parameter λ and gets the chameleon hash evaluation key *ek* from its challenger. Next, it initializes adversary $\mathcal{A}$ with the same security parameter λ. In this weaker security model, adversary $\mathcal{B}$ does not need to pass the signature public key yet.

- **Query:** Adversary $\mathcal{A}$ makes its single query sending the message msg_1. Now adversary $\mathcal{B}$ must send as response the signature public key and the signature for message msg_1.

  It chooses as signature for msg_1 some $rnd_1 \in R$ chosen uniformly at random. Next, it computes a suitable public key for the signature computing $dgt_1 \leftarrow CH.\text{HASH}(ek, msg_1, rnd_1)$. It sends to $\mathcal{A}$ the public key $pk = (ek, dgt_1)$ and the signature rnd_1.

  Notice that if the chameleon hash *CH* has the uniformity property, even if msg_1 was not chosen uniformly at random, this do not change the probability distribution of dgt_1 from what is expected in a legitimate key generation.

- **Finalization:** After receiving the response for the query and the public key, adversary $\mathcal{A}$ produces a forgery (msg', rnd'). If this is a correct forgery, this means that $CH.\text{HASH}(ek, msg', rnd') = dgt_1$ and $(msg', rnd') \neq (msg_1, rnd_1)$. Adversary $\mathcal{B}$ can output (msg_1, rnd_1) and (msg', rnd') as a collision in *CH*.

The adversary $\mathcal{B}$ always succeeds if the adversary $\mathcal{A}$ succeeds at finding a forgery. Therefore, for all adversaries $\mathcal{A}$, we can build efficient adversary $\mathcal{B}$ such that:

$$GCMAadv_{1\text{-}Weak}[\mathcal{A}, SIG_1] \leq CRadv[\mathcal{B}, CH]$$

Therefore, if the chameleon hash is collision-resistant, the right side of this inequation is always a negligible value. Therefore, all adversaries that forge signatures for SIG_1 also can succeed only with negligible probability. ■

5.2 ONE-TIME SIGNATURE FULLY SECURE AGAINST CHOSEN MESSAGE ATTACKS

Combining more than one application of chameleon hash we can build one-time signatures secure in a more rigorous security definition. The construction in this section, like SIG_1, is presented at (MOHASSEL, 2010).

We will use a collision-resistant chameleon hash CH defined over sets (M, R, D) and a collision resistant function $HASH : D \to M'$ with $M' \subset M$. We will define an one-time signature denoted by SIG_{OT} defined over sets $(M, R \times R)$. Assume that $x \in M$ is a known and public constant. The signature scheme algorithms are:

- SIG_{OT}.KEYGEN(λ):
  1. $(ek_0, tk_0) \xleftarrow{\$} CH$.KEYGEN($\lambda$)
  2. $(ek_1, tk_1) \xleftarrow{\$} CH$.KEYGEN($\lambda$)]
  3. $rnd_0 \xleftarrow{\$} R$
  4. $rnd_1 \xleftarrow{\$} R$
  5. $dgt_k \leftarrow CH$.HASH$(ek_0, 0, rnd_0)$
  6. $msg_k \leftarrow HASH(CH$.HASH$(ek_1, 0, rnd_1))$
  7. $pk \leftarrow (ek_0, ek_1, dgt_k)$
  8. $sk \leftarrow (tk_0, tk_1, rnd_0, rnd_1, msg_k)$
  9. **return** (pk, sk)
- SIG_{OT}.SIGN(sk, msg):
  1. $(tk_0, tk_1, rnd_0, rnd_1, msg_k) \leftarrow sk$
  2. $rnd'_0 \xleftarrow{\$} CH$.COLLISION$(tk_0, x, rnd_0, msg_k)$
  3. $rnd'_1 \xleftarrow{\$} CH$.COLLISION$(tk_1, x, rnd_1, msg)$
  4. $sig \leftarrow (rnd'_0, rnd'_1)$
  5. **return** sig
- SIG_{OT}.VERIFY(pk, msg, sig):
  1. $(ek_0, ek_1, dgt_k) \leftarrow pk$
  2. $(rnd'_0, rnd'_1) \leftarrow sig$
  3. $msg' \leftarrow$ HASH$(CH$.HASH$(ek_1, msg, rnd'_1))$
  4. $dgt' \leftarrow CH$.HASH(ek_0, msg', rnd'_0)
  5. **if** $dgt' = dgt_k$:
  6. **return** `accept`
  7. **else**:
  8. **return** `reject`

This signature scheme works because if the signature was created by SIG_{OT}.SIGN, then in the verification algorithm above, at line 3, the result of CH.HASH(ek_1, msg, rnd'_1) will be equal CH.HASH(ek_1, x, rnd_1) (because rnd'_1 was produced using CH.COLLISION). Therefore, the msg' obtained at that line will be equal msg_k. And in line 4 of the verification algorithm, using rnd'_0, the result of the chameleon hash will be equal CH.HASH$(ek_0, x, rnd_0) = dgt_k$.

Theorem 21 *If the chameleon hash CH and the function HASH are collision-resistant, then the one-time signature SIG_{OT} defined above is secure against chosen message attacks.*

Proof: We will assume that exist an efficient adversary $\mathcal{A}$ that forges signatures in SIG_{OT} and show hot to use $\mathcal{A}$ to build a new adversary $\mathcal{B}$ that finds collisions in CH.

Recall that adversary $\mathcal{A}$ will make a single signing query, as SIG_{OT} is one-time signature. We will call msg_1 the message sent in the signing query and will call (rnd_{01}, rnd_{11}) the response sent by $\mathcal{B}$. Assume that after this, adversary $\mathcal{A}$ sends the forgery $(msg', (rnd'_0, rnd'_1))$.

Any adversary $\mathcal{A}$ will act using one of the following two mutually exclusive strategies:

- It will produce a forgery such that $(msg', rnd'_1) \neq (msg_1, rnd_{11})$ and, at the same time, CH.HASH$(ek_1, msg_1, rnd_{11}) = CH$.HASH$(ek_1, msg', rnd'_1)$. We will denote by $\mathcal{A}_1$ any adversary that use this strategy.
- It will produce a forgery such that $(msg', rnd'_1) = (msg_1, rnd_{11})$, or such that CH.HASH$(ek_1, msg_1, rnd_{11}) \neq CH$.HASH$(ek_1, msg', rnd'_1)$. We will denote by $\mathcal{A}_2$ any adversary that use this strategy.

First we will produce an adversary $\mathcal{B}_1$ that finds collisions using some $\mathcal{A}_1$. Next, we will produce an adversary $\mathcal{B}_2$ that finds collisions in CH using adversary $\mathcal{A}_2$. In the end we assume the general case in which we do not know what strategy the adversary will use.

Our adversary $\mathcal{B}_1$ acts as below:

- **Initialization:** Adversary $\mathcal{B}_1$ is initialized with the security parameter λ and with a public key ek sent by its challenger. This key will be used as ek_1 in the signature scheme. Now it gets $(ek_0, tk_0) \xleftarrow{\$} CH$.KEYGEN$(\lambda)$ and produces $dgt_k = CH$.HASH(ek_0, x, rnd_0) for some rnd_0 chosen uniformly at random. It initializes adversary $\mathcal{A}_1$ sending $pk = (ek_0, ek_1, dgt_k)$.
- **Signing Query:** When $\mathcal{A}_1$ sends the first query msg_1, $\mathcal{B}_1$ computes the signature choosing some rnd_{11} uniformly at random and computing the correct

matching rnd_{01} as the result of $CH.\text{COLLISION}(tk_0, x, rnd_0, msg_k)$ for $msg_k = HASH(CH.\text{HASH}(ek_1, msg_1, rnd_{11}))$. Adversary $\mathcal{B}_1$ sends (rnd_{01}, rnd_{11}) as response to $\mathcal{A}_1$.

- **Finalization:** Adversary $\mathcal{A}_1$ produces a forgery $(msg', (rnd'_0, rnd'_1))$ such that $(msg', rnd'_1) \neq (msg_1, rnd_{11})$ and at the same time $CH.\text{HASH}(ek_1, msg_1, rnd_{11}) = CH.\text{HASH}(ek_1, msg', rnd'_1)$. Therefore, we have a collision in the chameleon hash using the key $ek_1 = ek$. Adversary $\mathcal{B}_1$ outputs collision $((msg_1, rnd_{11}), (msg', rnd'_1))$ and exits.

In this case, adversary $\mathcal{B}_1$ always succeeds when adversary $\mathcal{B}_1$ succeeds. We have:

$$CRadv[\mathcal{B}_1, CH] \geq CMAadv_1[\mathcal{A}_1, SIG_{OT}]$$

Now we define how our adversary $\mathcal{B}_2$ works:

- **Initialization:** Adversary $\mathcal{B}_2$ is initialized with the security parameter λ and with a public key ek. This key will be used as ek_0 in the signature scheme. After this it computes $(ek_1, tk_1) \xleftarrow{\$} CH.\text{KEYGEN}(\lambda)$, $msg_k = hash(CH.\text{HASH}(ek_1, x, rnd_1))$ and $dgt_k = CH.\text{HASH}(ek_0, x, rnd_0)$ for some (rnd_0, rnd_1) chosen uniformly at random. Adversary $\mathcal{A}_2$ is initialized with $pk = (ek_0, ek_1, dgt_k)$.
- **Signing Query:** When $\mathcal{A}_2$ sends the query msg_1, $\mathcal{B}_2$ computes the signature choosing some rnd_{01} uniformly at random and computing the correct matching rnd_{11} as the result of $CH.\text{COLLISION}(tk_1, 0, rnd_1, msg)$. Adversary $\mathcal{B}_2$ sends (rnd_{01}, rnd_{11}) as response to $\mathcal{A}_2$.
- **Finalization:** Adversary $\mathcal{A}_2$ produces a forgery $(msg', (rnd'_0, rnd'_1))$ such that either $(msg', rnd'_1) = (msg_1, rnd_{11})$ or otherwise, $CH.\text{HASH}(ek_1, msg_1, rnd_{11}) \neq CH.\text{HASH}(ek_1, msg', rnd'_1)$. In either case, we found with high probability a collision in the chameleon hash in the key $ek_0 = ek$ that result in the same digest dgt_k.

  The first value is obtained using the values sent in the query. We know that:

$$CH.\text{HASH}(ek_0, HASH(CH.\text{HASH}(ek_1, msg_1, rnd_{11})), rnd_{01}) = dgt_k$$

  The second value is obtained using the forgery:

$$CH.\text{HASH}(ek_0, HASH(CH.\text{HASH}(ek_1, msg', rnd'_1)), rnd'_0) = dgt_k$$

  This yields a collision in the chameleon hash with high probability. If $\mathcal{A}$ wins the attack game and $(msg', rnd'_1) = (msg_1, rnd_{11})$, this always produces a collision,

as this means that $rnd'_0 \neq rnd_{01}$. Otherwise, if $CH.\text{HASH}(ek_1, msg_1, rnd_{11}) \neq CH.\text{HASH}(ek_1, msg', rnd'_1)$, then this almost always produces a collision, except in the case where $rnd'_0 = rnd_{01}$ and the function *HASH* outputs the same value in the two equations above. However, if *HASH* is collision-resistant, this is unlikely.

The worst case for adversary $\mathcal{B}_2$ is precisely when $rnd'_0 = rnd_{01}$ because there is a possibility that $\mathcal{A}_2$ succeeds without returning useful information to produce a collision because it produced a collision in *HASH*. Therefore, the probability of $\mathcal{B}_2$ succeed is given by the probability of $\mathcal{A}_2$ succeed minus the probability of finding a collision in *HASH* during the interaction:

$$CRadv[\mathcal{B}_2, CH] \geq CMAadv_1[\mathcal{A}_1, SIG_{OT}] - CRadv[\mathcal{B}', HASH]$$

Finally, let's build an adversary $\mathcal{B}$ that finds a collision in *CH* without knowing if adversary $\mathcal{A}$ will behave like adversary $\mathcal{A}_1$ or like adversary $\mathcal{A}_2$. Adversary $\mathcal{B}$ can simply toss a coin to decide if it will behave like $\mathcal{B}_1$ or $\mathcal{B}_2$. The probability of guessing correctly is 1/2. Therefore, the probability of adversary $\mathcal{B}$ producing a collision is given by:

$$CRadv[\mathcal{B}, CH] \geq \frac{1}{2}(CMAadv_1[\mathcal{A}_1, SIG_{OT}] - CRadv[\mathcal{B}', HASH])$$

Therefore, for all adversaries $\mathcal{A}$, we can build adversaries $\mathcal{B}$ and $\mathcal{B}'$ such that:

$$CMAadv_1[\mathcal{A}_1, SIG_{OT}] \leq 2 \cdot CRadv[\mathcal{B}, CH] + CRadv[\mathcal{B}', HASH]$$

By our assumption, *CH* and *HASH* are collision-resistant. Therefore, the probability of creating a forgery against this one-time signature is always bounded by a negligible value. ■

5.3 SIGNATURE SCHEME SECURE AGAINST CLASSICAL ADVERSARIES

This section will present a new signature scheme strongly based on the ring signatures presented in (LU et al., 2019c). That paper presented two ring signatures, a concept described for the first time at (RIVEST et al., 2001).

In a ring signature, any person can sign a message on behalf of a group in which she belongs. In the verification process, the verifier can check that the signature was created by someone in that group, but it is not possible to guess who signed with a probability greater than $1/n$ where n is the number of people in the group. Contrary to the concept of "group signatures", in a ring signature it is not necessary to have a central authority to create groups, any person can create a new group only knowing the public keys of other participants.

However, the article (LU et al., 2019c) as published, presented no security proofs. An extended version was after published online and a proof for one of the ring signatures was presented. Some issues with the proof were then found and presented at

(WANG, Xueli et al., 2019). In this later paper, the author pointed the issues and proposed to solve them with another construction without using chameleon hash functions. Therefore, the construction presented at (LU et al., 2019c) remain as unproved to the best of our knowledge.

In this section we build a new scheme modifying the construction from (LU et al., 2019c) in a way that we can prove the security of the construction using the weakest assumptions that we could find. The result is a regular signature scheme (not a ring signature) that can be proven secure against classical attackers, but not against post-quantum attackers. Despite this being the best proof that we found, this does not discard the possibility that a better proof with less restricted conditions can be found. Or that the original ring signature scheme can be proven secure using other techniques.

First we will define the signature scheme SIG_Σ with the help of a chameleon hash CH defined over (M, R, D) and with a hash function $HASH : (M' \times \{0,1\}^k \times D) \to M$. Here the set $\{0,1\}^k$ represents the set of possible public keys and the number k is the key length. Notice that the digest space of the function $HASH$ is equal to the message space of the chameleon hash. The signature will be defined over sets $(M', M \times R)$. The signature algorithms are:

- SIG_Σ.KEYGEN(λ):
  1. $(ek, tk) \xleftarrow{\$} CH$.KEYGEN($\lambda$)
  2. $pk \leftarrow ek$
  3. $sk \leftarrow tk$
  4. **return** (pk, sk)
- SIG_Σ.SIGN(sk, msg):
  1. $(ek, tk) \leftarrow sk$
  2. $d \xleftarrow{\$} D$
  3. $m \leftarrow HASH(msg, ek, d)$
  4. $r \xleftarrow{\$} CH$.PREIMAGE($tk, m, d$)
  5. **return** (m, r)
- SIG_Σ.VERIFY(pk, msg, sig):
  1. $ek \leftarrow pk$
  2. $(m, r) \leftarrow sig$
  3. **if** $HASH(msg, ek, CH.\text{HASH}(ek, m, r)) = m$:
     a) **return** `accept`

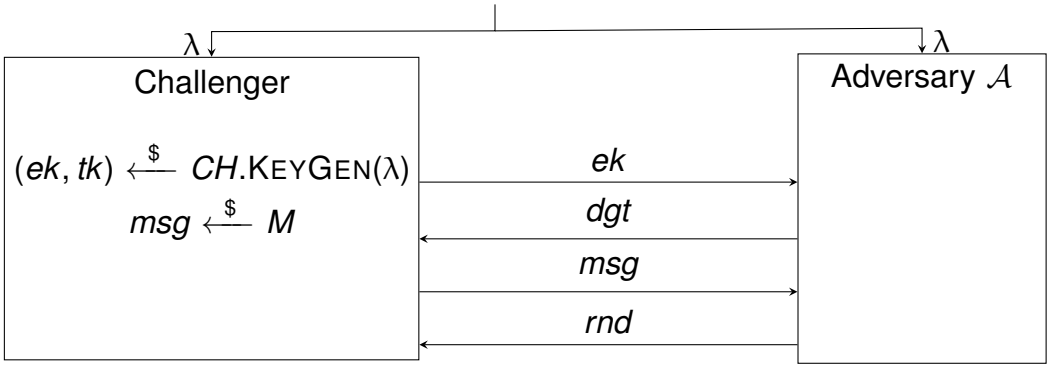

Figure 24 – Attack Game: Sigma Security

4. **else**:

   a) **return** `reject`

The reason for representing this signature scheme as SIG_Σ is because its security depends on the chameleon hash being secure in an alternative attack game, not the usual collision-resistance game. This alternative security definition can be seen as an attack against a Σ-protocol:

Attack Game 15 ***(Sigma-Secure).*** *For a chameleon hash CH defined over* (M, R, D)*, we model a game with an adversary and a challenger. Both are initialized by a security parameter* λ*.*

The challenger computes a pair of keys with $CH.\text{KEYGEN}(\lambda)$ *and sends the evaluation key ek to the adversary.*

The adversary chooses some $dgt \in D$ *(not necessarily random) and sends to the challenger. The challenger chooses a random and uniform* $msg \in M$ *and sends to the adversary as a challenge. The adversary sends some* $rnd \in R$ *as response.*

We say that the adversary wins this game if $CH.\text{HASH}(ek, msg, rnd) = dgt$*. For a given adversary* $\mathcal{A}$ *and chameleon hash CH, we denote as SIGMAadv*$[\mathcal{A}, CH]$ *the probability that the adversary wins this game.*

Definition 7 *A chameleon hash CH is* ***sigma-secure*** *if for all efficient adversaries* $\mathcal{A}$*, the value of SIGMAadv*$[\mathcal{A}, CH]$ *is negligible.*

If a chameleon hash is sigma-secure in the above definition, it means that for this chameleon hash, if the keys were generated using the standard algorithm $CH.\text{KEYGEN}$, the owner of the trapdoor can prove its ownership without revealing it by the exchange of three messages with a verifier. It chooses and sends some digest (that can be found computing a chameleon hash for a random pair of message and random parameter), gets a message as challenge and then use the algorithm $CH.\text{COLLISION}$ to produce the final response. The owner of the trapdoor always succeeds in this challenge, but other parties not, except with negligible probability.

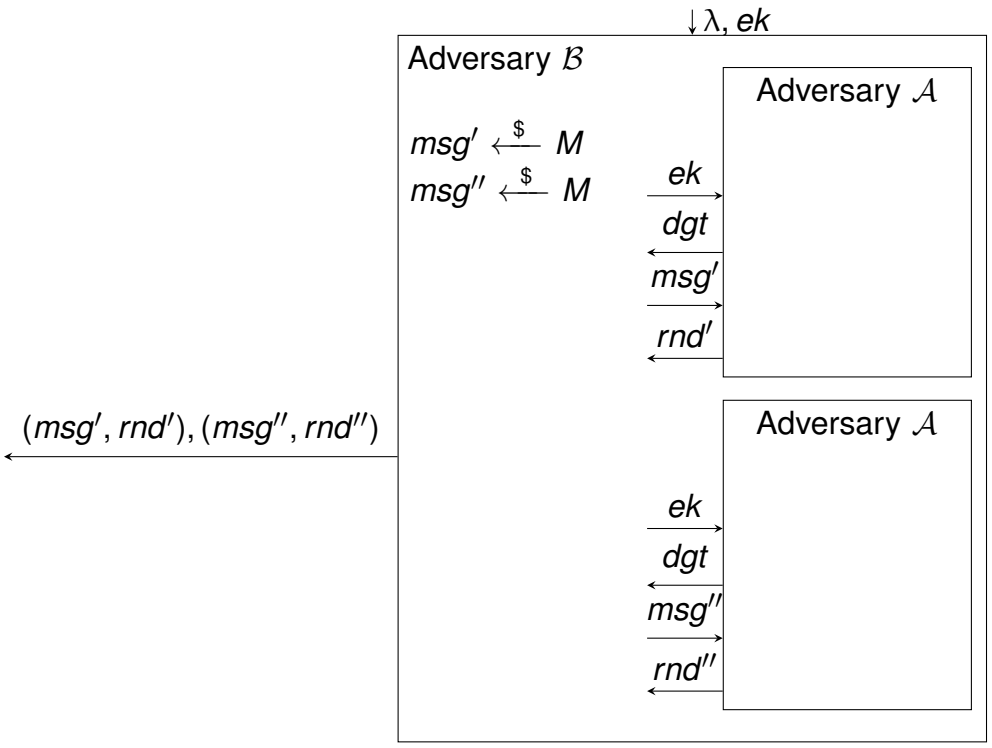

Figure 25 – Building a collision-finder $\mathcal{B}$ from attacker $\mathcal{A}$ against the sigma-security.

The next theorem was noted first in (BELLARE; RISTOV, 2014). The difference is that this article assume a classical model of computation, so do not explicitly mention quantum adversaries in the theorem:

Theorem 22 *All collision-resistant chameleon hash CH are sigma-secure against classical adversaries (but not necessarily against quantum adversaries).*

Proof: Let's assume that we have a classical adversary $\mathcal{A}$ that attacks the sigma-security of a chameleon hash and wins with non-negligible probability. Therefore, we can build an adversary $\mathcal{B}$ that finds a collision as in the following image:

In the above adversary $\mathcal{B}$, it runs two copies of adversary $\mathcal{A}$ and they are initialized with the same evaluation key *ek* and also the same security parameter λ. If $\mathcal{A}$ is a deterministic adversary, both copies will output the same digest *dgt*. If they are probabilistic adversaries, then they must be feed with exactly the same randomness source to produce the same digest *dgt*.

Once this happens, we send different messages as a challenge for both copies. Receiving different inputs, now we expect that both copies acts differently and sends different rnd' and rnd'' in the final output.

The adversary $\mathcal{B}$ succeeds if the following events happen:

- The first copy of $\mathcal{A}$ wins the attack game for sigma-security. This happens with probability $SIGMAadv[\mathcal{A}, CH]$.

- The second copy of $\mathcal{A}$ wins the attack game for sigma-security. This also also happens with probability $SIGMAadv[\mathcal{A}, CH]$.
- $(msg', rnd') \neq (msg'', rnd'')$. We can ensure that this is true if $msg' \neq msg''$. This happens with probability $(1 - \frac{1}{|M|})$. Therefore, the probability for this event is at least $(|M|-1)/|M|$. We can assume that $|M| \geq 2$, which means that this probability is always greater or equal than 1/2.

With this information, we can deduce this lower bound for the success of adversary $\mathcal{B}$:

$$CRadv[\mathcal{B}, CH] \geq \frac{1}{2}\,(SIGMAadv[\mathcal{A}, CH])^2$$

Using this lower bound for the success of $\mathcal{B}$, we can conclude that for all efficient adversaries $\mathcal{A}$ that attack the sigma-security of the chameleon hash CH, we can build an efficient adversary $\mathcal{B}$ that tries to find collisions in CH such that:

$$SIGMAadv[\mathcal{A}, CH] \leq \sqrt{2}(CRadv[\mathcal{B}, CH])^{1/2}$$

By our assumption, the chameleon hash CH is collision-resistant. Therefore, $CRadv[\mathcal{B}, CH]$ is always negligible. Therefore, the square root of this value multiplied by $\sqrt{2}$ also is negligible. And this proves that any efficient adversary $\mathcal{A}$ succeeds only with negligible probability while attacking the sigma-security.

■

The above proof uses what is called a "**rewinding argument**": running two instances of $\mathcal{A}$ while being able to feed them with the same randomness source is equivalent to have the ability to run $\mathcal{A}$ once, and then rewind it to a previous state and resume the interaction making different choices. This rewinding argument is possible using a classical model of computation. However, it cannot be done if we assume that $\mathcal{A}$ is a post-quantum adversary that is running quantum algorithms.

The reason is that by the models of quantum mechanics, it is impossible to clone or copy an arbitrary unknown quantum state, a limitation that is known as the "no-cloning theorem". We cannot have two exact copies of $\mathcal{A}$ with exactly the same state if $\mathcal{A}$ is in a quantum state. In quantum mechanics model, even if we run two quantum computers representing adversaries $\mathcal{A}$ under exactly the same physical conditions, we will not receive always the same results from $\mathcal{A}$. Both adversaries could return different values dgt and with this behaviour, we cannot use them to find collisions in CH. A quantum adversary that breaks the sigma-security apparently presents no contradiction with the collision-resistance property.

The sigma security is important in the signature scheme SIG_Σ of this section because if we have a chameleon hash sigma-secure, then the signature SIG_Σ built from it is secure against chosen message attacks in the random oracle model. As we can

guarantee sigma-security only against non-quantum adversaries, the signature SIG_Σ can be proven only in a classical model of computation.

Theorem 23 *If we have a collision-resistant function HASH and a sigma-secure chameleon hash CH with the uniformity property, then the signature scheme SIG_Σ defined in this section and built using CH and HASH is strongly secure against adaptively chosen message attacks in the random oracle model.*

Proof: Let's assume that we have an adversary $\mathcal{A}$ that forges a signature in the attack game for signature security against chosen-message attacks. Then we will use it to build a new adversary $\mathcal{B}$ that breaks the sigma-security of the chameleon hash.

In this proof, instead of build the entire adversary $\mathcal{B}$, we will start considering only an specific adversary $\mathcal{A}_0$ instead of a generic one. And we will first build a $\mathcal{B}_0$ that assumes that $\mathcal{A}_0$ uses an specific strategy to create a forgery. With this we can first focus in the main idea of our proof. After this we will begin to generalize our adversary showing how we can deal with other possible strategies that $\mathcal{A}$ can use. In the end we will produce an adversary $\mathcal{B}$ that can use any forger $\mathcal{A}$ to break the sigma-security.

Our first adversary $\mathcal{B}_0$ assumes that the adversary $\mathcal{A}_0$ produces a forgery after making one random oracle query and without making signing queries. It also assumes that $\mathcal{A}_0$ uses this random oracle query to produce the forgery. For example, it can first check that the result of $HASH(msg', (ek, d')) = m'$. And after discovering this, $\mathcal{A}_0$ can use some weakness in the chameleon hash CH to find a suitable r' such that $CH.\text{HASH}(ek, m', r') = d'$. Notice how in this case the pair (m', r') is a valid signature for msg' and how the problem of making this forgery is similar to attack the sigma-security game.

In this scenario, the adversary $\mathcal{B}_0$ proceeds as below:

- **Initialization:** The adversary $\mathcal{B}_0$ is initialized with the security parameter λ and the evaluation key ek. Then, the adversary $\mathcal{B}_0$ initializes the adversary $\mathcal{A}_0$ passing the value ek as the signature public key.
- **Random Oracle Query:** Receiving the random oracle query $(msg', (ek, d'))$, the adversary $\mathcal{B}_0$ starts to interact with its challenger sending d'. The challenger, as required by the sigma-security game chooses a random and uniform $m' \in M$ and send to $\mathcal{B}_0$. As m' was chosen uniformly at random, we can use it as the response for the random oracle query. We send m' to $\mathcal{A}_0$.
- **Finalization:** The adversary $\mathcal{A}_0$ produces a forgery $(msg', (m', r'))$. Assuming that this forgery was produced using the random oracle query, this msg' is the same message queried in the random oracle query and this m' is the same value sent to $\mathcal{B}_0$ by its challenger. Therefore, if this is a forgery, then $CH.\text{HASH}(ek, m', r') = d'$ and $\mathcal{B}_0$ can win its game sending r' to its challenger.

Notice how $\mathcal{B}_0$ always wins its game if $\mathcal{A}_0$ succeeds at producing a forgery.

However, not all possible adversaries $\mathcal{A}$ will act this way. For example, consider an adversary $\mathcal{A}_1$ that makes no random oracle query and no signature query. For example, an adversary that tries to make a forgery choosing a signature at random. We will now build an adversary $\mathcal{B}_1$ that breaks the sigma-security of *CH* with the same probability than $\mathcal{A}_1$ succeeds.

During initialization, adversary $\mathcal{B}_1$ acts exactly as adversary $\mathcal{B}_1$, passing the chameleon hash key as the signature key to $\mathcal{A}_1$. There are no random oracle or signing queries. The main difference is how $\mathcal{B}_1$ acts during the finalization:

- **Finalization:** Adversary $\mathcal{A}_1$ produces a forgery $(msg', (m', r'))$. Now adversary $\mathcal{B}_1$ proceeds running the verification algorithm SIG_Σ.VERIFY. To do so, it first computes $d' = CH.\text{HASH}(ek, m', r')$ and sends d' to its challenger. The challenger sends as response a m' chosen uniformly at random. Adversary $\mathcal{B}_1$ sets m' as the result of the random oracle for input $(msg', (ek, d'))$. If the signature sent by $\mathcal{A}_1$ is valid, then $CH.\text{HASH}(ek, m', r') = d'$. Therefore, $\mathcal{B}_1$ can win the sigma-security game in this case sending r' to its challenger.

Again the adversary $\mathcal{B}_1$ wins its game with exactly the same probability that $\mathcal{A}_1$ produces a forgery.

Now let's generalize our scenario assuming that we have an adversary $\mathcal{A}_2$ that makes one random oracle query, no signatures queries, but we do not know if it will use the random oracle query to produce a forgery like adversary $\mathcal{A}_0$ or if the forgery produced in the end has no direct relationship with some previous random oracle query, as is the case with adversary $\mathcal{A}_1$.

In this scenario, our adversary $\mathcal{B}_2$ can act in its game in the following way:

- **Initialization:** Choose a random bit $b \xleftarrow{\$} \{0, 1\}$. If 0 was chosen, then $\mathcal{B}_2$ will act as adversary $\mathcal{B}_0$. If $b = 1$, then it will act more like adversary $\mathcal{B}_1$. Now adversary $\mathcal{B}_2$ initializes adversary $\mathcal{A}_2$ sending the public key *ek* for it.

- **Random Oracle Query:** If we are acting like adversary $\mathcal{B}_0$, we follow exactly what was described for this adversary during its query. We use the value presented in the query to interact with $\mathcal{B}_2$'s challenger and use the response to set the response for the random oracle. If we are acting more like adversary $\mathcal{B}_1$, we assume that this random oracle query will not help us in breaking the sigma-security. We just choose uniformly at random a response $m_1 \in M$ and send this as the response for the query.

- **Finalization:** We use exactly the same finalization for adversary $\mathcal{B}_0$ or adversary $\mathcal{B}_1$, depending of which adversary we are imitating.

Now we can see that to win the game, adversary $\mathcal{B}_2$ must choose the right bit b depending of $\mathcal{A}_2$ strategy. The lower bound for the success of $\mathcal{B}_2$ is given by:

$$SIGMAadv[\mathcal{B}_2, CH] \geq \frac{1}{2} CMAadv^{RO}[\mathcal{A}_2, SIG_\Sigma]$$

Let's keep generalizing the model assuming that now we have an adversary $\mathcal{A}_3$ that still makes no signing queries, but can make a polynomial number of q_O random oracle queries. The value q_O is bounded by a polynomial over the security parameter λ.

Now the adversary $\mathcal{B}_3$ instead of choosing a random bit, must choose a value uniformly at random between 1 and $q_O + 1$. And need to store the random oracle values to keep the responses consistent if the same value is queried more that once:

- **Initialization:** Choose a random value $q' \xleftarrow{\$} \{0, \ldots, q_O + 1\}$. Next initialize an dictionary to keep all random oracle queries and their respective responses. The dictionary is initialized empty. Now initialize a counter q with 0. This counter will increment each time we get a new random oracle query. And proceed the initialization as the previous adversaries, initializing adversary $\mathcal{A}_3$ with the public key ek.

- **Random Oracle Query:** For each query, adversary $\mathcal{B}_3$ first check if this query was already done before. For this, it checks in the dictionary if the query is stored there and if this is true, it sends exactly the same response stored there. If the query never was done before, it increments the counter q. If $q = q'$, $\mathcal{B}_3$ acts in this query exactly as adversary $\mathcal{B}_0$: it assumes that this query will be used in the forgery. It uses the queried value to interact with the challenger and use the challenger's response as the response for the random oracle query. Otherwise, if $q \neq q'$, then $\mathcal{B}_3$ produces a response choosing a random and uniform $m_i \in M$. In either case, when we produce a response for the random oracle query, we store the query and the response as a pair in our dictionary. Acting this way, $\mathcal{B}_3$ keeps the consistence of its random oracle simulation.

- **Finalization:** If $q' = q_O + 1$, adversary $\mathcal{B}_3$ surely did not interacted with its challenger. In this case, it acts exactly as adversary $\mathcal{B}_1$ in the finalization using the signature verification to interact with its challenger. Otherwise, it acts exactly as adversary $\mathcal{B}_0$ assuming that we guessed correctly which query was used in the forgery.

Notice that adversary $\mathcal{A}_3$ can be any possible adversary that makes no signing queries. For any adversary $\mathcal{A}$ that makes no signing queries, the final forgery is or is not produced using a previous random oracle query. There are no other third possibility.

Now given this adversary $\mathcal{B}_3$, we can deduce its lower bound for success as:

$$SIGMAadv[\mathcal{B}_3, CH] \geq \frac{1}{q_O + 1} CMAadv^{RO}[\mathcal{A}_3, SIG_\Sigma]$$

Finally, we can finish our generalization producing the final adversary $\mathcal{B}$ that can deal with any adversary $\mathcal{A}$. We not only let adversary $\mathcal{A}$ make a polynomially bounded number of q_O random oracle queries, but we also let it make a polynomially bounded number of q_S signing queries. All queries are adaptive and can be done in any order.

In this case, we can use an adversary $\mathcal{B}$ that acts exactly as adversary $\mathcal{B}_3$. The sole difference is that now it needs to produce responses for signing queries. It can deal with such queries in the following way:

- **Signing Query:** The adversary $\mathcal{A}$ sends a message msg_i to be signed. Our adversary $\mathcal{B}$ do not know the secret key for the signature SIG_Σ. However, using the fact that it controls the random oracle simulation, it can produce valid signatures with overwhelming probability. First $\mathcal{B}$ generates random and uniform values $m_i \in M$ and $r_i \in R$ and set this random pair (m_i, r_i) as the signature. To make this signature valid, adversary $\mathcal{B}$ also computes $d_i = CH.\text{HASH}(ek, m_i, r_i)$. And knowing the digest d_i, it sets m_i as the result for random oracle query for $(msg_i, (ek, d_i))$. Adversary $\mathcal{B}$ stores this in the dictionary used by our random oracle simulator. This means that after this, future random oracle queries for $(msg_i, (ek, d_i))$ will always return m_i. After all these steps, $\mathcal{B}$ sends (m_i, r_i) to $\mathcal{A}$ as response for the query.

Notice that if the chameleon hash CH has the uniformity property, the value d_i generated computing $CH.\text{HASH}(ek, m_i, r_i)$ for a random r_i is equivalent to choosing uniformly at random some value from digest space D, exactly as is required for signature creation by our definition of SIG_Σ.

Also notice that the value m_i was chosen uniformly at random from M and that M is also the digest space for $HASH$. Therefore, the above strategy correctly choose a random and uniform value for the random oracle query $(msg_i, (ek, d_i))$ as is required to simulate a random oracle.

However, there is a problem in how our adversary $\mathcal{B}$ answers the signing query. What if some value other than m_i was already associated with $(msg_i, (ek, d_i))$? Perhaps a random oracle query was already produced for $(msg_i, (ek, d_i))$. Or a previous signing query for the same message msg_i produced a distinct signature (m_j, r_j) for $m_j \neq m_i$ such that $CH.\text{HASH}(ek, m_i, r_i) = CH.\text{HASH}(ek, m_j, r_j)$. If this happens, adversary $\mathcal{B}$ cannot at the same time produce a correct signature and keep the consistency of the simulation. We assume that in this case it halts, losing the attack game against the sigma-security.

What would be the probability that adversary $\mathcal{B}$ fails to produce a signature in the above signature query? We can find an upper bound for a single signing query considering that in the worst case we already produced q_O different random oracle

results computed during random oracle queries and also another $q_s - 1$ random oracle values produced in previous signing queries. We are performing the last permitted signing query. We also assume that all the queries were for the same message $msg \in M'$ and for the public key ek. In this worst-case scenario for a single query, we have a probability of $\frac{q_O+q_S-1}{|D|}$ of producing a new digest d_i in our signing query such that $\mathcal{B}$ fails at producing a signature. This assumes that the digest d_i produced during the signing query is random and uniform, as consequence of the uniformity property.

Given this upper bound for a single query, we can multiply it by the number of signing queries to obtain an upper bound for the probability of $\mathcal{A}$ failing while computing all signing queries. And the complement of this value, $\left(1 - \frac{q_s(q_O+q_S-1)}{|D|}\right)$, is the lower bound for the probability of $\mathcal{B}$ succeed in all signing queries.

Finally, we can conclude that adversary $\mathcal{B}$ succeeds if:

1. Adversary $\mathcal{A}$ succeeds.
2. Adversary $\mathcal{B}$, exactly like adversary $\mathcal{B}_3$ guessed the correct value for q choosing a number between 1 and $q_O + 1$.
3. Adversary $\mathcal{A}$ succeeds producing responses for all signing queries.

Using the probabilities of the above three events, we can produce the following lower bound for the success of $\mathcal{B}$:

$$SIGMAadv[\mathcal{B}, CH] \geq \left(1 - \frac{q_s(q_O + q_S - 1)}{|D|}\right) \frac{1}{q_O + 1} CMAadv^{RO}[\mathcal{A}, SIG_\Sigma]$$

As this last adversary $\mathcal{B}$ can deal with any adversary $\mathcal{A}$ that produces forgeries, we can rewrite the above inequation concluding that for all efficient adversaries $\mathcal{A}$, we can build efficient adversaries $\mathcal{B}$ such that:

$$CMAadv^{RO}[\mathcal{A}, CH] \leq (q_O + 1) \left(\frac{|D|}{|D| - q_S(q_O + q_S)}\right) SIGMAadv[\mathcal{B}, CH]$$

In the above inequation, the first term of the multiplication in the right side is polynomially bounded, as q_O is polynomially bounded. The second term is negligibly next to 1, as $|D|$ is superpolynomial and both q_O and q_S are bounded polynomially. Finally, the last term of the multiplication is always negligible, because by our assumption, CH is sigma-secure. This means that the right-side of the inequation is always negligible because it is the multiplication of a negligible value by polynomially bounded values. And this proves that the signature is secure in the random oracle model. ■

As we do not have a proof that any chameleon hash is sigma-secure against quantum adversaries, this result is more useful for chameleon hash functions in the classical model of computation, ignoring quantum adversaries. In this model we can

rewrite the above formula using the relationship between sigma-security and collision-resistance, obtaining the result that for all $\mathcal{A}$, we can obtain $\mathcal{B}$ such that:

$$CMAadv^{RO}[\mathcal{A}, CH] \leq (q_O + 1)\left(\frac{|D|}{|D| - q_S(q_O + q_S)}\right)\sqrt{2}(CRadv[\mathcal{B}, CH])^{1/2}$$

5.4 BUILDING UNIVERSAL DESIGNATED SIGNATURES

This signature construction was proposed for the first time at (LI, B. et al., 2018) and combines a specific signature scheme (the ring variant of the GPV signature that we will define in this section) with a collision-resistant chameleon hash with preimage uniformity property to produce what is known as a "universal designated signature".

An universal designated signature is a special case of signature scheme where we have not only the algorithms *SIG*.KEYGEN, *SIG*.SIGN and *SIG*.VERIFY, but also three other additional algorithms in the scheme:

- *SIG*.DESIGNATE(pk_s, pk_r, *msg*, *sig*): It takes as input a message *msg*, its respective signature *sig*, the signer public key pk_s and some recipient public key pk_r. It produces a new signature *dsig* designated specifically to the recipient associated with the public key pk_r.
- *SIG*.DESIGNATEDVERIFY(pk_s, pk_r, *msg*, *dsig*): It gets as input a signer public key, a recipient public key, some message and a designated signature. It returns `accept` or `reject`.
- *SIG*.SIMULATE(sk_r, pk_s, *msg*): This algorithm produces a forged designated signature *dsig* for the message *msg* attributed to the signer whose public key is pk_s to the recipient whose secret key if sk_s.

The idea of this scheme is that if we have a designated signature *dsig*, we cannot differentiate between a legitimate signature created using the algorithm *SIG*.DESIGNATE or a forgery created using *SIG*.SIMULATE if we are no the designated recipient. However, if we are the designated recipient, we can know if a signature is legitimate. As only we know our secret key sk_r, if we never produced this designated signature using *SIG*.SIMULATE, then it must be a legitimate signature. Despite knowing this, we cannot prove for other people the legitimacy of a signature.

Like the chameleon signatures presented in Subsection 3.4.1, this scheme is intended to be used when we want to sign a message for a single recipient, but perhaps because the content of a message is confidential and delicate, we do not want that the signature proves the authenticity of the message for third parties. Contrary to the chameleon signatures, the scheme also can produce traditional signatures and any person can generate a designated signatures given a legitimate signature (as this

involves only public keys). The disadvantage is that this scheme as presented does not offer a method to solve disputes.

To show how universal designated signatures could be built from chameleon hash functions, first we will present a regular signature scheme known as the Ring-GPV signature, and then we will extend this regular signature to a universal designated signature with the help of the chameleon hash CH_{SIS}. Before defining the signature, we will show how the algorithms TRAPGEN, SAMPLEPRE and SAMPLEDOM specified in Subsection 4.2.2 can be adapted to the polynomial ring setting.

5.4.1 Ring Version of TRAPGEN, SAMPLEPRE and SAMPLEDOM

First recall the algorithms TRAPGEN, SAMPLEPRE and SAMPLEDOM specified in Subsection 4.2.2. We can adapt all these algorithms to the Ring-SIS assumption making them act over vectors of elements in a polynomial ring as required by the Ring-SIS assumption.

Given the description of a modular polynomial ring R and the polynomial ring R_q where all the coefficients are modulo q, and given parameters m, q associated with Ring-SIS assumption and the distribution $\mathcal{D}$, our ring-version of the TRAPGEN algorithm is:

- RINGTRAPGEN($R_q, m, n, q, \mathcal{D}$):
  1. $\bar{\mathbf{a}} \xleftarrow{\$} R_q^{m-w}$
  2. $\mathbf{T} \xleftarrow{\mathcal{D}} R_q^{(m-w)\times(n \lg q)}$
  3. $\mathbf{a} \leftarrow \bar{\mathbf{a}} || (\mathbf{g} - \bar{\mathbf{a}}\mathbf{T})$
  4. **return** $(\mathbf{a}, \mathbf{T})$

Here we also assume that we produced a vector $\mathbf{a}$ indistinguishable from a random and uniform vector from R_q^m. In other words, we assume that the function below is a suitable weak pseudo-random function assuming any constant $\mathbf{g}$ which is a vector with w elements:

$$PRF_{\text{RINGTRAPGEN}}(\mathbf{T}, \mathbf{a}) = \mathbf{g} - \bar{\mathbf{a}}\mathbf{T})$$

Using this construction, we also have the property that:

$$\mathbf{a} \begin{bmatrix} \mathbf{T} \\ \mathbf{I} \end{bmatrix} = (\bar{\mathbf{a}} || \mathbf{g} - \bar{\mathbf{a}}\mathbf{T}) \begin{bmatrix} \mathbf{T} \\ \mathbf{I} \end{bmatrix} = \bar{\mathbf{a}}\mathbf{T} + \mathbf{g} - \bar{\mathbf{a}}\mathbf{T} = \mathbf{g}$$

Therefore, we can adapt directly the algorithm SAMPLEPRE to the ring setting. It will function in the same way, but recall that in this version $y()$ is a polynomial from R_q:

- RINGSAMPLEPRE($R_q, m, q, \mathbf{a}, \mathbf{T}, y()$):

1. $\mathbf{p} \xleftarrow{\mathcal{D}} R^m$
2. $v() \leftarrow y() - \mathbf{ap} \mod q$
3. Find a small $\mathbf{z}$ such that $\mathbf{gz} = v() \mod q$
4. $\mathbf{x} \leftarrow \mathbf{p} + \begin{bmatrix} \mathbf{T} \\ \mathbf{I} \end{bmatrix} \mathbf{z}$
5. If $||\mathbf{x}||$ is not small enough, return to step 1
6. **return x**

We assume that $\mathbf{g}$ is a vector for which is easy to find short vectors $\mathbf{x}$ such that $\mathbf{gx}$ is any desired result. We could use a vector $\mathbf{g}$ using an analogous method from Subsection 4.2.2.2 where we could build $\mathbf{x}$ choosing as coefficients values 0 and 1 forming the binary representation for coefficients in the desired result.

Exactly like in the SIS version of the construction, when we use these functions usually we use with parameters (R_q, m, q) and with a maximum allowed size such that lots of different valid results exist for the RINGSAMPLEPRE algorithm. We have a chance of choosing any of the possible solutions depending on the random $\mathbf{p}$ that we choose in line 1. When we use this algorithm we will assume that no solution have a probability greater that 1/2 of being chosen and that we have at least two different possible results for any value $y()$ passed to this algorithm.

Finally, we will also assume the existence of algorithm RINGSAMPLEDOM that given as input $(R_q, m, n, q, \mathbf{a})$, return a vector $\mathbf{x}$ with a short euclidean norm such that the probability distribution for the output is the same than what would be obtained choosing a random and uniform polynomial $y()$ and running RINGSAMPLEPRE$(R_q, m, n, q, \mathbf{a}, \mathbf{T}, y())$.

If in some construction we expect that RINGSAMPLEPRE and RINGSAMPLEDOM return vectors with norm smaller than a given value, then we will express this maximum norm as index during the invocation of these algorithms.

5.4.2 The Ring-GPV Signature Scheme

The GPV signature was first proposed at (GENTRY et al., 2008) and had its security based on the SIS assumption. Here we will need a ring-variant of that signature and we define it with the help of RINGTRAPGEN, RINGSAMPLEPRE and RINGSAMPLEDOM algorithms.

This version of the GPV signature scheme is defined over (M, S) and uses a hash function $HASH_1 : M \rightarrow D$ such that $D \subseteq R_q$. The set of signatures is $S \subset R_q^m$ such that the norm of a given signature is smaller than $\beta/2$.

The algorithms of the scheme are defined as below:

- SIG_{GPV}.KEYGEN(λ):

1. $(R_q, m, n, q, \beta) \xleftarrow{\$} \text{RSISPARAMS}(\lambda)$
2. $(\mathbf{a}, \mathbf{T}) \xleftarrow{\$} \text{RINGTRAPGEN}(\lambda)$
3. $pk \leftarrow (R_q, m, q, \beta, \mathbf{a})$
4. $sk \leftarrow (R_q, m, q, \beta, \mathbf{a}, \mathbf{T})$
5. **return** (pk, sk)

- SIG_{GPV}.SIGN(sk, msg)
  1. $(R_q, m, q, \beta, \mathbf{a}, \mathbf{T}) \leftarrow sk$
  2. $y() \leftarrow \text{HASH}_1(msg)$
  3. $\mathbf{s} \xleftarrow{\$} \text{RINGSAMPLEPRE}_\beta/2(R_q, m, q, \mathbf{a}, \mathbf{T}, y())$
  4. **return s**
- SIG_{GPV}.VERIFY$(pk, msg, \mathbf{s})$:
  1. $(R_q, m, q, \beta, \mathbf{a}) \leftarrow pk$
  2. **if** $||\mathbf{s}|| < \beta$ **and** $\mathbf{a} \cdot \mathbf{s} = \text{HASH}_1(msg)$:
  3. **return** `accept`
  4. **else**:
  5. **return** `reject`

There is also an additional requirement for the security of the above signature. There should always have more than one possible valid signature for any given message and if we sign the same message more than once, we should always produce the same signature. The first requirement is easily achieved generating correct parameters using RSISPARAMS. The later requirement can be fulfilled memorizing each signature or deriving all the probabilistic decisions for algorithm RINGSAMPLEPRE from a pseudo-random generator fed with a fixed seed chosen during key generation.

Assuming these two requirements above, we present the security proof of this signature below:

Theorem 24 *If the Ring-SIS assumption is true, the vector* ***a*** *returned by* RINGTRAPGEN *is indistinguishable from a random and uniform vector, if there are always more than one valid signature for each message such that no valid signature and if we set SIG_{GPV} to never return different signatures for the same message, then SIG_{GPV} is a secure signature scheme against adaptive chosen-message attacks in the random oracle model.*

Proof: To prove this, we will, as usual, build an efficient adversary $\mathcal{B}$ that solves the Ring-SIS attack game with non-negligible probability assuming that we have an

efficient adversary $\mathcal{A}$ that produces a forgery against signature SIG_{GPV} in the adaptive chosen message attack game and in the random oracle model.

Adversary $\mathcal{B}$ proceeds as below:

- **Initialization:** Adversary $\mathcal{B}$ is initialized with the security parameter λ and receives from its challenger a random and uniform vector $\mathbf{a} \in R_q^m$ along the parameters (R_q, m, q, β). It initializes a dictionary that is empty in the beginning, but will store pairs of messages and signatures. Finally, it initializes adversary $\mathcal{A}$ with the public key $(R_q, m, q, \beta, \mathbf{a})$.

- **Random Oracle Query:** For each random oracle query, adversary $\mathcal{B}$ gets some message msg_i sent by $\mathcal{A}$. First it checks if the message msg_i is in the dictionary. If so, a signature $\mathbf{s}_i$ for msg_i was already produced. We send to $\mathcal{A}$ the value $\mathbf{a} \cdot \mathbf{s}_i$ as response. By definition, if $\mathbf{s}_i$ is a correct signature, this scalar multiplication produces the correct hash or random oracle for msg_i.

  If msg_i is not in the dictionary, msg_i never was queried before in any query. Then first we will choose a valid signature for msg_i and using the signature we will deduce the correct response for the random oracle query. We choose the signature running RINGSAMPLEDOM$(R_q, m, n, q, \mathbf{a})$. Recall that by definition this will result in a vector with probability distribution analogous to computing RINGSAMPLEPRE over a random and uniform $y()$. Therefore, choosing the signature in this way simulates correctly a signature generated legitimately by SIG_{GPV}.SIGN. And after choosing the signature $\mathbf{s}_i$, we store it in the dictionary and send $\mathbf{as}_i$ as response for the random oracle query.

- **Signing Query:** For each signing query, $\mathcal{B}$ receives from **A** a message msg_i. First it checks if the message msg_i is in the dictionary. If so, it returns the signature stored with the message. If not, $\mathcal{B}$ produces a new signature running $\mathbf{s}_i \xleftarrow{\$}$ RINGSAMPLEDOM$(R_q, m, n, q, \mathbf{a})$, stores the pair $(msg_i, \mathbf{s}_i)$ in the dictionary and sends $\mathbf{s}_i$ to $\mathcal{A}$.

- **Finalization:** After $\mathcal{A}$ outputs a forgery $(msg', \mathbf{s'})$, adversary $\mathcal{A}$ recovers from the dictionary the signature **s"** associated with the message msg'. If msg' is not in the dictionary, we act as described above in the random oracle query to discover the result of the random oracle $\mathcal{O}(msg')$.

  In both cases, if $\mathcal{A}$ produced a correct forgery, we end with two correct signatures associated with msg': the signature **s'** produced by $\mathcal{A}$ and the signature **s"** stored in the dictionary kept by $\mathcal{B}$ when it used msg' in a previous query.

  Therefore, we have that $\mathbf{a} \cdot \mathbf{s'} = \mathbf{as"}$. Assuming that $\mathbf{s'} \neq \mathbf{s"}$, we have that $A(\mathbf{s}' - \mathbf{s}'') \bmod q = 0$. Adversary $\mathcal{B}$ outputs $(\mathbf{s}' - \mathbf{s}'') \bmod q$. The result is correct because

if both signatures have norm smaller than $\beta/2$, the returned value has a norm smaller than β.

Notice that adversary $\mathcal{B}$ always wins its attack game if:

1. $\mathcal{A}$ succeeds. This happens with probability $CMAadv^{RO}[\mathcal{A}, SIG_{GPV}]$.
2. $\mathbf{s} \neq \mathbf{s}''$. If $\mathcal{A}$ queried during the game a signing query for msg', this happens with probability 1, as by the rules $\mathcal{A}$ would need a different signature to succeed.

   If $\mathcal{A}$ never queried a signing query for msg', then exist a possibility that **s'** = **s"**. However, as we discussed in the previous subsection when described algorithm RIGSAMPLEPRE, we can safely assume that no possible valid signature for msg' has a probability greater than 1/2 of being chosen. The lower bound for the probability Pr[**s'** = **s"**] is therefore 1/2.

If we assume that adversary $\mathcal{A}$ cannot distinguish between a real random and uniform vector **a** and a vector returned by RINGTRAPGEN, then in this case the lower bound for the success of $\mathcal{B}$ is:

$$RSISadv[\mathcal{B}] \geq \frac{1}{2} CMAadv^{RO}[\mathcal{A}, SIG_{GPV}]$$

However, if we consider that adversary $\mathcal{A}$ can also create forgeries exploring some weakness in algorithm RINGTRAPGEN, the real lower bound is less than this. We need to subtract the probability that $\mathcal{A}$ succeed exploring such weakness distinguishing between a vector returned by RINGTRAPGEN and a real random and uniform vector. This probability can be modeler by $WPRFadv[\mathcal{B}', PRF_{\text{RINGTRAPGEN}}]$ for some adversary $\mathcal{B}$. Therefore, the real lower bound is:

$$RSISadv[\mathcal{B}] \geq \frac{1}{2} CMAadv^{RO}[\mathcal{A}, SIG_{GPV}] - WPRFadv[\mathcal{B}', PRF_{\text{RINGTRAPGEN}}]$$

Rewriting the above inequation, we have that for all efficient adversaries $\mathcal{A}$, we can build efficient adversaries $\mathcal{B}$ and $\mathcal{B}'$ such that:

$$CMAadv^{RO}[\mathcal{A}, SIG_{GPV}] \leq 2RSISadv[\mathcal{B}] + WPRFadv[\mathcal{B}', PRF_{\text{RINGTRAPGEN}}]$$

As by our assumption, the Ring-SIS problem is hard and the vector returned by RINGTRAPGEN is indistinguishable from a random and uniform vector, the right side of the above equation is negligible. Therefore, no adversary can create a forgery in the GPV signature with non-negligible probability. ■

This is an example of security proof using the random oracle model where we simulate the random oracle in a way that we can produce for each query a response in an homogeneous way: no single query is treated differently, we did not try to encode any specific information in the response for these queries and we do not use the input

in any query to guess something about the forgery that will be produced. This contrast with the sample security proof in the random oracle model from Subsection 2.4.1.1 where we tried to guess which query would be used in the forgery and treat differently this query inserting specific information in the response.

This homogeneous treatment gives us a more tight security proof, but other than that, it also makes this proof adaptable to a post-quantum scenario. From Subsection 2.4.1.2 the only requisite that this proof does not satisfy is the third one, of efficient simulation: using the method presented above of simulating a random oracle with a dictionary, we cannot efficiently evaluate the random oracle over a superpolynomial superposition of messages. As described in that subsection, in such cases it is possible to adapt the proof for quantum adversaries at the cost of an additional assumption.

If we use a pseudo-random function to generate the random values necessary during the queries and such function can be evaluated over quantum inputs in superposition and is secure even in this scenario (it is a quantum-accessible secure *PRF*), then we could use this *PRF* to generate responses in the queries and we could also use it to produce consistent responses even when a superposition of a superpolynomial number of messages is sent in a random oracle query. The additional assumption that the proof needs in a post-quantum scenario is the existence of such quantum-accessible *PRF*.

5.4.3 Extending the GPV Signature with Chameleon Hash Functions

To create a universal designated verifier signature *DSIG*, we can begin with the GPV signature defined over (M', S), using the same algorithms for signing and verifying. Instead of requiring that our signatures have a euclidean norm smaller than $\beta/2$ as required in the GPV signature, we will now require a norm smaller than $\frac{\beta}{4\sqrt{n}(\sqrt{n}+1)}$.

We will also require that our polynomial ring R_q be defined modulo $(x^n - 1)$ where n is power of 2 and q is a prime congruent to 5 (modulo 8). As discussed in (LYUBASHEVSKY; NEVEN, Gregory, 2017), this creates an interesting property where while not all elements in R_q have multiplicative inverses, we can guarantee that all elements with a short euclidean norm have inverses.

We use a chameleon hash defined over (M, R, D) to create our algorithms *DSIG*.DESIGNATE, *DSIG*.DESIGNATEDVERIFY and *DSIG*.SIMULATE. The message space of this chameleon hash is $M = R_q$.

We also use in this part a new collision-resistant hash function $HASH_2 : R_q^m \times D \times M' \rightarrow C$. Here the set C is the set of polynomials in R_q such that its coefficients are 0 or 1.

A designated signature will be a tuple from $C \times Z \times R$ such that Z is a subset of vectors form Z_q^m such that the euclidean norm of them is smaller than $\beta/4\sqrt{n}$.

We omit the *DSIG*.SIGN and *DSIG*.VERIFY algorithms, as they are practically identical to the algorithms in SIG_{GPV}, only requiring shorter signatures. The remaining

algorithms for the designated verifier signature are defined as below:

- *DSIG*.KEYGEN(λ):
  1. $(pk, sk) \xleftarrow{\$} SIG_{GPV}$.KEYGEN($\lambda$)
  2. $(ek, tk) \xleftarrow{\$} CH$.KEYGEN($\lambda$)
  3. **return** (pk' = (pk, ek), sk' = (sk, ek, tk))
- *DSIG*.DESIGNATE($pk_s, pk_v, msg, \mathbf{s}$):
  1. $((R_q, m, q, \beta, \mathbf{a}_s), ek_s) \leftarrow pk_s$
  2. $((R_q, m, q, \beta, \mathbf{a}_v), ek_s) \leftarrow pk_v$
  3. $\mathbf{x} \xleftarrow{\$}$ RINGSAMPLEDOM$_{\beta/(4\sqrt{n}(\sqrt{n}+1))}(R_q, m, n, q, \mathbf{a}_s)$
  4. $y() \leftarrow \mathbf{a}_s \cdot \mathbf{x}$
  5. $rnd \xleftarrow{\$} R$
  6. $dgt \leftarrow CH$.HASH$(ek_v, y(), rnd)$
  7. $c() \leftarrow HASH_2(\mathbf{a}_s, dgt, msg)$
  8. $\mathbf{z} \leftarrow \mathbf{s}c + \mathbf{x}$
  9. **return** $(c(), \mathbf{z}, rnd)$
- *DSIG*.DESIGNATEDVERIFY($pk_s, pk_v, msg, (c(), \mathbf{z}, rnd)$):
  1. $((R_q, m, q, \beta, \mathbf{a}_s), ek_s) \leftarrow pk_s$
  2. $((R_q, m, q, \beta, \mathbf{a}_v), ek_s) \leftarrow pk_v$
  3. $y'() \leftarrow \mathbf{a}_s \cdot \mathbf{z} - HASH_1(msg)c()$
  4. $dgt' \leftarrow CH.Hash(ek_v, y'(), rnd)$
  5. $c'() \leftarrow HASH_2(\mathbf{a}_s, dgt', msg)$
  6. **if** $||\mathbf{z}|| < \beta/4\sqrt{n}$ **and** $c() = c'()$:
  7. **return** `accept`
  8. **else**:
  9. **return** `reject`
- *DSIG*.SIMULATE(pk_s, sk_v, msg):
  1. $((R_q, m, q, \beta, \mathbf{a}_s), ek_s) \leftarrow pk_s$
  2. $((R_q, m, q, \beta, \mathbf{a}_v, \mathbf{T}_v), tk_v) \leftarrow sk_v$
  3. $y() \xleftarrow{\$} R_q$

4. $rnd \xleftarrow{\$} R$
5. $dgt \leftarrow CH.\text{HASH}(ek_V, y, rnd)$
6. $c() \leftarrow HASH_2(\mathbf{a}_S, dgt, msg)$
7. $\mathbf{s} \xleftarrow{\$} \text{RINGSAMPLEDOM}_{\beta/(4\sqrt{n}(\sqrt{n}+1))}(R_q, m, n, q, \mathbf{a}_S)$
8. $\mathbf{x} \xleftarrow{\$} \text{RINGSAMPLEDOM}_{\beta/(4\sqrt{n}(\sqrt{n}+1))}(R_q, m, n, q, \mathbf{a}_S)$
9. $\mathbf{z} \leftarrow \mathbf{s}c() + \mathbf{x}$
10. $y'() \leftarrow \mathbf{a}_S \cdot \mathbf{z} - HASH_1(msg)c()$
11. $rnd' \xleftarrow{\$} CH.\text{PREIMAGE}(tk_V, y'(), dgt)$
12. **return** $(c(), \mathbf{z}, rnd')$

Notice that we always use the signer's GPV keys in the above algorithms, never its chameleon hash keys. For the verifier we always use the chameleon hash keys, not the GPV keys. If we know that a person only signs messages but never will be the recipient of a designated signature, we could generate only GPV keys for her. Likewise, if an user never will sign messages, but could be the recipient of a designated signature, this person needs only the chameleon hash keys.

In algorithm *DSIG*.DESIGNATEDVERIFY we check in line 6 the requirement that $\mathbf{z}$ must have an euclidean norm smaller than $\beta/4\sqrt{n}$. To check that designated signatures produced by *DSIG*.DESIGNATE really produce vectors $\mathbf{z}$ with such characteristic, note that such value is produced at line 8 of such algorithm. And both $\mathbf{s}$ and $\mathbf{x}$ have euclidean norm smaller than $\beta/(4\sqrt{n}(\sqrt{n}+1))$. Recall that $c()$ is a polynomial whose coefficients are always 0 or 1 and therefore has at most $\sqrt{n}$ as norm. Therefore, $||\mathbf{s}c()|| < \frac{\beta\sqrt{n}}{4\sqrt{n}(\sqrt{n}+1)}$ and after the sum with $\mathbf{y}$, the upper bound for the norm became $\beta/4\sqrt{n}$.

The algorithm *DSIG*.DESIGNATEDVERIFY works because at line 3 it recovers exactly the same polynomial $y'()$ produced in line 4 of algorithm *DSIG*.DESIGNATE. This happens because the signature for message *msg* is the vector $\mathbf{s}$ such that $\mathbf{a}_S \cdot \mathbf{s} = HASH_1(msg)$ and so:

$$\begin{aligned}\mathbf{a}_S \cdot \mathbf{z} - HASH_1(msg)c() &= \mathbf{a}_S(\mathbf{s}c() + \mathbf{x}) - HASH_1(msg)c() \\ &= \mathbf{a}_S\mathbf{s}c() + \mathbf{a}_S\mathbf{x} - \mathbf{a}_S\mathbf{s}c() \\ &= \mathbf{a}_S\mathbf{x} \mod q\end{aligned}$$

Therefore, algorithm *DSIG*.DESIGNATEDVERIFY, if used in a legitimate signature, will produce in line 5 exactly the same result than algorithm *DSIG*.DESIGNATE produces in line 7.

Algorithm *DSIG*.SIMULATE also produces accepted designated signatures, but does so computing first a random chameleon hash digest in line 5, using this digest to

produce $c()$ and $\mathbf{z}$, and finally, use the algorithm *CH*.PREIMAGE to produce a valid rnd' such that the correct input for the chameleon hash during the verification produces the expected digest.

5.4.4 Security of the Universal Designated Verifier Signature

The security of universal designated verifier signatures is given by two properties. The first is that it is not possible to distinguish a designated signature produced using *DSIG*.SIMULATE and a designated signature produced using *DSIG*.DESIGNATE from a valid signature previously produced with *DSIG*.SIGN. This is the privacy property and can be proven if the chameleon hash *CH* has preimage uniformity.

Theorem 25 *If the preimage chameleon hash CH has the property of uniformity and preimage uniformity, then the universal designated signature DSIG produced with it has indistinguishable designated signatures when they are produced by DSIG*.SIMULATE *or DSIG*.DESIGNATE*. .*

Proof: A designated signature is a tuple $(c(), \mathbf{z}, rnd)$. We will show how for each of them we cannot identify if it was produced from *DSIG*.SIMULATE or *DSIG*.DESIGNATE, except with a probability negligibly next to 1/2.

The value $c()$, both in *DSIG*.SIMULATE and in *DSIG*.DESIGNATE is produced computing $HASH_2$. The first and third input for this hash function is the same in both cases. The second parameter are in both cases produced computing a chameleon hash choosing the random parameter uniformly at random. By the uniformity property, these two inputs cannot be distinguished, and therefore, this is also true for the value $c()$.

The vector $\mathbf{z}$ in algorithm *DSIG*.DESIGNATE is chosen as $\mathbf{s}c() + \mathbf{y}$ where both $\mathbf{y}$ and $\mathbf{s}$ are small vectors and $\mathbf{s}$ is a GPV signature produced with RINGSAMPLEPRE. In algorithm *DSIG*.SIMULATE, the vector z is produced with $\mathbf{s}'c() + \mathbf{y}'$ with the sole difference that here $\mathbf{s}'$ is produced with RINGSAMPLEDOM. However, the algorithms RINGSAMPLEDOM and RINGSAMPLEPRE produce indistinguishable results when algorithm SAMPLEPRE is computing the preimage of a random and uniform vector. Therefore, in both cases the vector $\mathbf{z}$ cannot be distinguished.

Finally, both in *DSIG*.DESIGNATE and in *DSIG*.SIMULATE, the third value of the result is a random parameter for the chameleon hash. In the first case, it is chosen uniformly at random. In the second, it is computed from *CH*.PREIMAGE given a random and uniform *dgt*. Therefore, its indistinguishability came from the preimage uniformity present in the chameleon hash. ■

Next, we need to define the concept of unforgeability for universal designated verifier signature. We will use the following attack game in the definition:

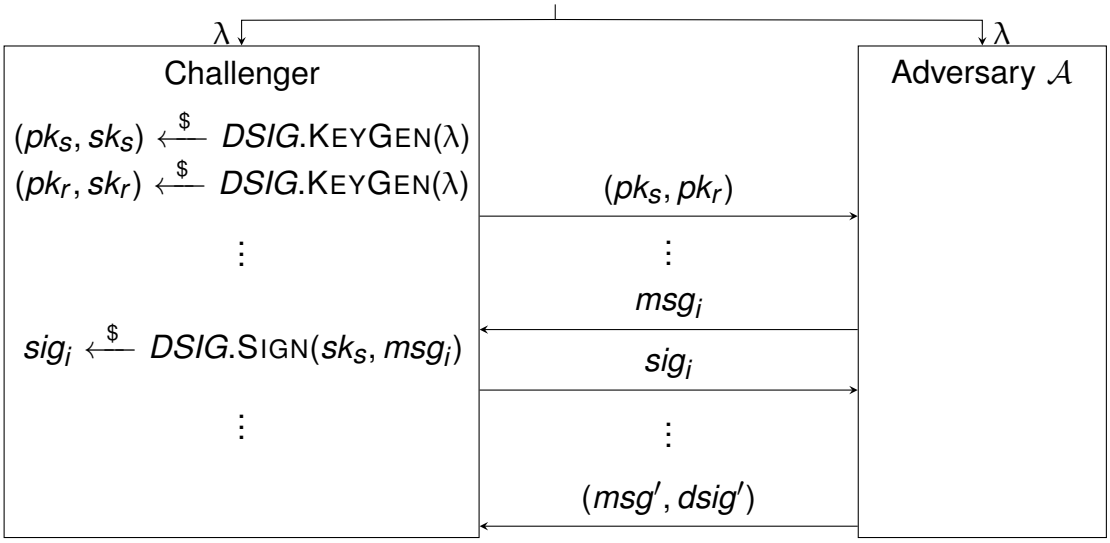

Figure 26 – Attack Game: Unforgeability against chosen message attacks for universal designated verifier signatures

Attack Game 16 ***(Universal Designated Verifier Signature Unforgeability against Adaptive Chosen Message Attack)****. The game is defined using a challenger and an adversary $\mathcal{A}$. Both are initialized using the security parameter λ. The challenger runs the signature key generation twice. The first time it produces a pair of keys (pk_s, sk_s) for a signer and the second time it produces a pair of keys (pk_r, sk_r) representing the keys for a recipient of designated signatures. After this, it sends the pair (pk_s, pk_r) for the adversary.*

Next, the adversary can make a polynomially bounded number of q_s signing queries. In each of them, it sends a message msg_i and gets as response its signature sig_i produced with the secret key sk_s.

After all the queries, the adversary produces as output a pair composed of a message msg' and a designated signature attributed to the user of the public key pk_s and designated to the user of public key pk_r. We say that it wins the game if the message msg' was not used in a signing query and DSIG.$\text{DESIGNATEDVERIFY}(pk_s, pk_r, msg', dsig')$ *outputs* `accept`*.*

We denote by DCMAadv$[\mathcal{A}, DSIG]$ *the probability of a given adversary $\mathcal{A}$ wins this game against the universal designated verifier signature scheme DSIG.*

Definition 8 *We say that an universal designated verifier signature DSIG is unforgeable against chosen message attacks, if for all adversaries $\mathcal{A}$, the value of the probability DCMAadv*$[\mathcal{A}, DSIG]$ *is negligible.*

Before proving the unforgeability of *DSIG*, recall that as discussed before presenting the algorithms *DSIG*.DESIGNATE, *DSIG*.DESIGNATEDVERIFY and *DSIG*.SIMULATE, this construction uses parameters such that all polynomials with short euclidean norm have multiplicative inverses in the polynomial ring R.

Notice that by definition, polynomials returned by $HASH_2$ are polynomials whose coefficients are always 0 or 1, therefore all of them have short euclidean norm and have multiplicative inverses. If we subtract two results of $HASH_2$, the result is a new polynomial whose coefficients are 0, 1 or -1. All of them also have inverse multiplicative. And with these coefficients, its inverse multiplicatives also have coefficients composed only by 0, 1 or -1 with norm at most $\sqrt{n}$. This means that if $c''()$ and $c'()$ are both the result of $HASH_2$, we have that $||c''() - c'()|| < \sqrt{n}$ and $||(c''() - c'())^{-1}|| < \sqrt{n}$.

In the security proof for *DSIG* we will use the random oracle model. Considering that *DSIG* uses two hash functions: $HASH_1$ (which is also used by the SIG_{GPV} signature scheme) and $HASH_2$, this means that we will use two random oracles denoted by $\mathcal{O}_1$ and $\mathcal{O}_2$.

Theorem 26 *If the chameleon hash CH is collision-resistant and SIG_{GPV} is unforgeable against adaptive chosen message attacks, then the designated verifier signature DSIG is unforgeable against adaptive chosen message attacks.*

Proof: We will create an efficient adversary $\mathcal{B}$ that breaks the unforgeability of SIG_{GPV} using an efficient adversary that forges a designated signature for *DSIG*. We will first consider a specific adversary $\mathcal{A}_1$ and build an adversary $\mathcal{B}_1$ that uses $\mathcal{A}_1$. Next, we will generalize this adversary with a new adversary $\mathcal{A}_2$ and show how we can adapt and build $\mathcal{B}_2$ using $\mathcal{A}_2$. Finally, after some generalizations, we will have an adversary $\mathcal{A}$ that encompass all possible adversaries and using it we will build the final version of our adversary $\mathcal{B}$.

First consider as $\mathcal{A}_1$ an adversary that creates a forgery for *DSIG* after performing signing queries and random oracle queries both for $\mathcal{O}_1$ and for $\mathcal{O}_2$. Moreover, adversary $\mathcal{A}_1$ will always create a forgery for some message msg' using the result of its last query for $\mathcal{O}_2$.

In this case, we can build adversary $\mathcal{B}_1$ in the following way:

- **Initialization:** Adversary $\mathcal{B}_1$ gets from its challenger the GPV public key pk composed of a vector of polynomials $\mathbf{a}_s \in R_q^m$ as well as all other parameters (m, q, β) associated with the Ring-SIS assumption. It runs the chameleon hash key generation algorithm to produce (ek_s, tk_s) to the signer. The signer public key is $pk_s = (pk, ek_s)$. The recipient public key pk_r can be produced running the algorithms SIG_{GPV}.KeyGen and CH.KeyGen. After generating all these keys, it initializes adversary $\mathcal{A}_1$ sending (pk_r, pk_s).

- **Random Oracle Query for $\mathcal{O}_1$:** When adversary $\mathcal{A}_1$ queries for the result of $\mathcal{O}_1(msg_i)$, adversary $\mathcal{B}_1$ forwards this query for its challenger and forwards the received response to $\mathcal{A}_1$.

- **Signing Query:** The signing queries set by $\mathcal{A}_1$ also are forwarded to $\mathcal{B}_1$'s challenger and the response is sent again to $\mathcal{A}_1$.

- **Random Oracle Query for $\mathcal{O}_2$:** When adversary $\mathcal{A}_1$ queries for the result of $\mathcal{O}_2(\mathbf{a}_i, dgt_i, msg_i)$, adversary $\mathcal{B}_1$ chooses the response $c_i() \in C$ uniformly at random and sends it as response to $\mathcal{A}_1$. It also stores this query and its response to send exactly the same value if the same query is sent again.

- **First Finalization:** In the end, adversary $\mathcal{A}_1$ outputs forgery $(msg', (c'(), \mathbf{z'}, rnd'))$. Adversary $\mathcal{B}_1$ store this result.

- **Rewinding:** Run adversary $\mathcal{A}_1$ a second time. Give it exactly the same input than before. Provided that it uses the same source of randomness, its queries will be the same. However, when it sends the last query for the second random oracle, the one used in the forgery, adversary $\mathcal{B}_1$ sends a new random and uniform value $c''()$.

- **Second Finalization:** We know that $\mathcal{A}_1$ always produces a forgery using the last random oracle query that represents $HASH_2$. We know that in both executions, the last random oracle query was the same tuple $(\mathbf{a}_S, dgt', msg')$. And both times the adversary $\mathcal{A}_1$ produced a designated signature for the same message msg'. However, with high probability, we sent different responses for this same random oracle query. In the first execution of $\mathcal{A}_1$, we sent some random and uniform value $c'()$ and the second time our response was $c''()$. This means that with high probability, adversary $\mathcal{A}_1$ produced two different designated signatures during its two executions: $(c'(), \mathbf{z}', rnd')$ and $(c''(), \mathbf{z}'', rnd'')$.

  Assuming that adversary $\mathcal{A}_1$ succeeds in creating the forgery both times, this means by the definition of algorithm *DSIG*.DESIGNATEDVERIFY, we have that $CH.\text{HASH}(ek_V, \mathbf{a}_S \cdot \mathbf{z'}) - \mathcal{O}_1(msg')c'(), rnd') = dgt'$ and that $CH.\text{HASH}(ek_V, \mathbf{a}_S \cdot \mathbf{z}'' - \mathcal{O}_1(msg')c''(), rnd'') = dgt'$. Therefore, either we found a collision in the chameleon hash or in both cases the chameleon hash had exactly the same input.

  If we had a collision in this chameleon hash, adversary $\mathcal{B}_1$ halts. Otherwise, if we did not found a collision, then this means that we can use the output to produce a forgery for SIG_{GPV}. In this case, we have that $rnd = rnd''$ and $\mathbf{a}_S \cdot \mathbf{z}' - \mathcal{O}_1(msg')c'() = \mathbf{a}_S \cdot \mathbf{z''} - \mathcal{O}_1(msg')c''()$. Rewriting this equation we have $\mathbf{a}_S \cdot (\mathbf{z}' - \mathbf{z}'') = \mathcal{O}_1(msg')(c'() - c''())$. As $(c'() - c''())$ is a vector with a short euclidean norm, it has a multiplicative inverse. And $(\mathbf{z}' - \mathbf{z}'')(c'() - c''())^{-1}$ is a valid GPV signature to msg'. Adversary $\mathcal{B}_1$ outputs msg' and the computed GPV signature as its output.

About the size of the output, notice that as $||F'|| < \beta/4\sqrt{n}$ and $||F''|| < \beta/4\sqrt{n}$. Therefore (**z'** $-$ **z"**) has norm smaller than $\beta/2\sqrt{n}$. And multiplying by $(c'() - c''())$ we are multiplying by a value with norm smaller than $\sqrt{n}$. Therefore:

$$||(\mathbf{z'} - \mathbf{z"})(c'() - c''())|| \leq \beta/2$$

If $\mathcal{A}_1$ produced a forgery with correct euclidean norm bounds, then $\mathcal{B}_1$ can produce a forgery in the GPV signature such that the forgery also has a sufficient short euclidean norm as required by the SIG_{GPV} signature.

The above adversary $\mathcal{B}_1$ succeeds if the following events happens:

1. Adversary $\mathcal{A}_1$ succeed the first time it is executed. This happens with probability $DCMAadv[\mathcal{A}, DSIG]$.
2. Adversary $\mathcal{A}_1$ succeed the second time it is executed. This also happens with probability $DCMAadv[\mathcal{A}, DSIG]$.
3. During both executions we sent different values $c'()$ and $c''()$ as response for the last query for $\mathcal{O}_2$. This happens with probability $(1 - \frac{1}{2^n})$.
4. The two forgeries do not produce a collision in the chameleon hash *CH*. The probability of finding a collision can be modelled as the probability of some adversary $\mathcal{B}'$ finding a collision in *CH* (this adversary $\mathcal{B}'$ can be built simulating the interaction between our adversaries and its challenger). Therefore, the probability of finding a collision instead of a possible forgery is given by $CRadv[\mathcal{B}', CH]$.

Notice that the first three events are independent while the last event is not. It is possible that adversary $\mathcal{A}_1$ succeeds only after finding a digest for *CH* such that it can easily find preimages or collisions. Therefore, to produce a correct lower bound for the probability of success for $\mathcal{B}_1$, we should compute the probability of producing in the second finalization a forgery or a collision, and then we subtract from this the probability of finding the collision:

$$CMAadv^{RO}[\mathcal{B}_1, SIG_{GPV}] \geq \left(1 - \frac{1}{2^n}\right)\left(DCMAadv^{RO}[\mathcal{A}_1, DSIG]\right)^2 - CRadv[\mathcal{B}', CH]$$

Next, let's generalize more our adversary considering an adversary $\mathcal{A}_2$ that creates a forgery in *DSIG* using the result of any query for $\mathcal{O}_2$, not necessarily the last one. Let's denote by q_{O2} the number of random oracle queries for $\mathcal{O}_2$. This means that each of these queries have a probability $1/q_{O2}$ of being used in the forgery.

To interact with this adversary $\mathcal{A}_1$, we can build an adversary $\mathcal{B}_2$ that acts exactly as adversary $\mathcal{B}_1$, except during the rewinding step. During this step, adversary $\mathcal{B}_2$ now acts as below:

- **Rewinding:** Interact again with adversary $\mathcal{A}_2$ after the end of the first interaction. Repeat until it sends again the query for $\mathcal{O}_2(\mathbf{a}_s, dgt', msg')$ used in the first interaction. Now instead of repeating exactly the same responses for $\mathcal{O}_2$ queries, produce new values uniformly at random.

Now to adversary $\mathcal{B}_2$ succeed, adversary $\mathcal{A}_2$ needs to choose during the second execution the same query chosen during the first execution to produce its forgery even considering that we changed the response for this query and for the queries after it.

This scenario is more complex because the event in which the adversary choose the same query twice not necessarily is independent from the event in which we sent different responses fo the same chosen input. And this also is not necessarily independent of the event in which the adversary $\mathcal{A}_2$ succeed when executing a second time.

This scenario can be modelled by a known theorem called the general forking lemma, present in (BELLARE; NEVEN, Greagory, 2006). The theorem description and proof is present in Appendix A. According with the general forking lemma, the probability that $\mathcal{A}_2$ succeeds in both executions, choose the same query twice and the response for the chosen query differ in both executions is at least:

$$DCMAadv^{RO}[\mathcal{A}_2, DSIG]\left(\frac{DCMAadv^{RO}[\mathcal{A}_2, DSIG]}{q_{O2}} - \frac{1}{2^n}\right)$$

In this case, adversary $\mathcal{B}_2$ either can compute a forgery for SIG_{GPV} or can find a collision in CH. The lower bound for the probability of creating a forgery in SIG_{GPV} is then:

$$CMAadv^{RO}[\mathcal{B}_1, SIG_{GPV}] \geq DCMAadv^{RO}[\mathcal{A}_2, DSIG]\left(\frac{DCMAadv^{RO}[\mathcal{A}_2, DSIG]}{q_{O2}} - \frac{1}{2^n}\right) - CRadv[\mathcal{B}', CH]$$

Now let's consider the most general adversary $\mathcal{A}$. It can produce a forgery using any queried result like $\mathcal{A}_2$, or it can produce a forgery without using directly any query for $\mathcal{O}_2$. To produce our final adversary $\mathcal{B}$, we use exactly the same construction than adversary $\mathcal{B}_2$. Except that now adversary $\mathcal{B}$, halts if in the first or in the second execution, adversary $\mathcal{A}$ produce a forgery that do not use information got from some query for $\mathcal{O}_2$.

Notice that in a correct forgery $(c(), \mathbf{z}, rnd)$ we need to evaluate $\mathcal{O}_2$ and check if the result is equal to $c()$. If this value never was queried, the probability of producing a correct forgery is always exactly $1/2^n$ because 2^n is the number of different values in the range of $\mathcal{O}_2$. Therefore, this is the probability that adversary $\mathcal{A}$ succeeds without using a query to the second random oracle. In this scenario, to produce our final lower bound for the success of $\mathcal{B}$, we will subtract $1/2^n$ from the probability of $\mathcal{A}$ succeed.

This subtraction removes the events in which adversary $\mathcal{A}$ succeed, without the help of a query for $\mathcal{O}_2$:

$$CMAadv^{RO}[\mathcal{B}, SIG_{GPV}] \geq \left(DCMAadv^{RO}[\mathcal{A}, DSIG] - \frac{1}{2^n} \right) \cdot \left(\frac{DCMAadv^{RO}[\mathcal{A}, DSIG] - frac12^n}{q_{O2}} - \frac{1}{2^n} \right) - CRadv[\mathcal{B}', CH]$$

To get a upper bond for $DCMAadv^{RO}[\mathcal{A}_2, DSIG]$, we can begin rewritting the above inequation as:

$$CMAadv^{RO}[\mathcal{B}_2, SIG_{GPV}] + CRadv[\mathcal{B}', CH] \geq \frac{(DCMAadv^{RO}[\mathcal{A}, DSIG] - \frac{1}{2^n})^2}{q_{O2}} - \frac{DCMAadv^{RO}[\mathcal{A}, DSIG] - \frac{1}{2^n}}{2^n}$$

Notice that independent of the value $DCMAadv^{RO}[\mathcal{A}, DSIG]$, the subtracted fraction in the right side of the above inequation is always a negligible value lesser or equal to $1/2^n$. We can replace this subtracted fraction in the formula above by $1/2^n$. And then, reorganizing the terms, we can conclude that for all adversaries $\mathcal{A}$, we can build adversaries $\mathcal{B}$ and $\mathcal{B}'$ such that:

$$DCMAadv^{RO}[\mathcal{A}, DSIG] \leq \sqrt{q_{O2}(CMAadv^{RO}[\mathcal{B}_2, SIG_{GPV}] + CRadv[\mathcal{B}', CH] + \frac{1}{2})} + \frac{1}{2^n}$$

We know that $1/2^n$ is negligible. If *CH* is collision-resistant and if SIG_{GPV} is secure against chosen message attacks, then both probabilities $CMAadv[\mathcal{B}_2, SIG_{GPV}]$ and $CRadv[\mathcal{B}', CH]$) are negligible and and so is the sum of these three elements. Multiplying a negligible value by the polynomially bounded q_{O2} we still get a negligible value and this means that the square root above also result in a negligible value. This means that all adversaries that can create forgeries using chosen message attacks against *DSIG* can succeed only with negligible probability. ■

Notice that we needed the rewinding argument to prove the security of this signature. Therefore, the above bound is proven only for classical algorithms. The proof and the bound are not valid against quantum algorithms, even if SIG_{GPV} is secure against such adversaries.

5.5 HOMOMORPHIC SIGNATURES WITH CHAMELEON HASH FUNCTIONS

The idea of using the homomorphism in some chameleon hash functions to build homomorphic signatures appear in (GORBUNOV et al., 2015). A homomorphic signature *HSIG* is defined over (M, S, C), being M the message space, S the signature

space and C a set of possible circuits that map M^k to M. The scheme is composed of four algorithms:

- *HSIG*.KEYGEN(λ, k): Takes as input a security parameter λ and an integer k. It returns a pair (pk, sk) where pk is a public key and sk is a secret key.
- *HSIG*.SIGN(sk, msg, i): Takes as input a secret key sk, message msg and an index $i \in \{1, \ldots, k\}$. It returns a signature sig_i.
- *HSIG*.EVAL($pk, \mathcal{C}, (msg_1, sig_1), \ldots (msg_k, sig_k)$): Takes as input a public key, the specification of some circuit $\mathcal{C}$ that maps k messages $(msg_1, \ldots, msg_k)$ to a single message msg' and a list of k pairs of messages and signatures. The algorithm outputs a new signature sig'.
- *HSIG*.VERIFY($pk, \mathcal{C}, msg, sig$): Takes as input a public key, the specification of some circuit $\mathcal{C} \in C$ and a pair of message and signature. It outputs `accept` or `reject`.

The scheme is correct if for all tuples of k messages, if we sign each one with a different index, and if we pass them with their signatures ordered by their index to *HSIG*.EVAL with any circuit $\mathcal{C} \in C$, we obtain a new signature sig' such that *HSIG*.VERIFY($pk, \mathcal{C}(msg_1, \ldots, msg_k), sig'$) = `accept`.

In other words, if we sign the input for some circuit, we can derive the correct signature for the output of the same circuit.

As an example, we can build the signature scheme *HSIG* defined over (M, S, C) using the lattice-based chameleon hash CH_{HTDF} from Subsection 4.2.3, defined over (M, R, D). Assume that for any $\mathcal{C} \in C$ we can build some $\mathcal{C}_R : R^k \to R$ and $\mathcal{C}_D : D^k \to D$ such that if for all $i \in [1, k]$ we have CH.HASH($ek, msg_i, rrnd_i$), then:

$$CH.\text{HASH}(ek, \mathcal{C}(msg_1, \ldots, msg_k), \mathcal{C}_R(rnd_1, \ldots, rnd_k)) = \mathcal{C}_D(dgt_1, \ldots, dgt_k)$$

Using the chameleon hash CH_{HTDF} and the above functions, we can buld *HSIG* in the following way:

- *HSIG*.KEYGEN(λ, k):
  1. $(ek, tk) \xleftarrow{\$} CH_{HTDF}.\text{KEYGEN}(\lambda)$
  2. $(\mathbf{D}_1, \ldots, \mathbf{D}_k) \xleftarrow{\$} D^k$
  3. $pk \leftarrow (ek, (\mathbf{D}_1, \ldots, \mathbf{D}_k))$
  4. $sk \leftarrow tk$
  5. **return** (pk, sk)
- *HSIG*.SIGN(sk, msg, i):

1. $tk \leftarrow sk$
2. **return** CH_{HTDF}.PREIMAGE(tk, msg, V_i)

- *HSIG*.EVAL$(pk, \mathcal{C}, (msg_1, sig_1), \ldots, (msg_k, sig_k))$:
  1. $(ek, (\mathbf{D}_1, \ldots, \mathbf{D}_k)) \leftarrow pk$
  2. Derive a corresponding $\mathcal{C}_R$ from $\mathcal{C}$
  3. **return** $\mathcal{C}_R(sig_1, \ldots, sig_k)$
- *HSIG*.VERIFY$(pk, \mathcal{C}, msg, sig)$:
  1. $(ek, (\mathbf{D}_1, \ldots, \mathbf{D}_k)) \leftarrow pk$
  2. Derive a corresponding $\mathcal{C}_D$ from $\mathcal{C}$
  3. **if** CH_{HTDF}.HASH$(ek, msg, sig) = \mathcal{C}_D(\mathbf{D}_1, \vdots, \mathbf{D}_k)$:
  4. **return** `accept`
  5. **else**:
  6. **return** `reject`

We still need to define what is the set C of algorithms supported in our homomorphic signature and how we derive suitable $\mathcal{C}_R$ and $\mathcal{C}_D$ given any circuit $\mathcal{C}$. We will define four different operations supported by algorithms in C:

- **Addition of messages:** Given two messages $m_1, m_2 \in M$, we can define the addition circuit $\mathcal{C}(m_1, m_2) = m_1 + m_2 \mod q$. Given this circuit, we can define $\mathcal{C}_R$ and $\mathcal{C}_D$ in the following way:

$$\mathcal{C}_R(\mathbf{R}_1, \mathbf{R}_2) = \mathbf{R}_1 + \mathbf{R}_2 \mod q \qquad \mathcal{C}_D(\mathbf{D}_1, \mathbf{D}_2) = \mathbf{D}_1 + \mathbf{D}_2$$

  The reason because it works is that if CH_{HTDF}.HASH$(ek, m_1, \mathbf{R}_1) = \mathbf{D}_1$ and if CH_{HTDF}.HASH$(ek, m_2, \mathbf{R}_2) = \mathbf{D}_2$, then by the definition of CH_{HTDF}:

$$\begin{aligned} CH_{HTDF}.\text{HASH}(ek, m_1 + m_2, \mathbf{R}_1 + \mathbf{R}_2) &= (m_1 + m_2)\mathbf{G} + \mathbf{A}(\mathbf{R}_1 + \mathbf{R}_2)) \\ &= m_1\mathbf{G} + m_2\mathbf{G} + \mathbf{AR}_+1 + \mathbf{AR}_2 \\ &= m_1\mathbf{G} + \mathbf{AR}_2 + m_2\mathbf{G} + \mathbf{AR}_2 \\ = CH_{HTDF}.\text{HASH}(ek, m_1, \mathbf{R}_1) + &\ CH_{HTDF}.\text{HASH}(ek, m_2, \mathbf{R}_2) \end{aligned}$$

  Notice that if $||\mathbf{R}_1||_\infty < \beta'$ and if $||\mathbf{R}_2||_\infty < \beta'$, then $||\mathbf{R}_1 + \mathbf{R}_2||_\infty < 2\beta'$. Therefore, to keep the result secure according with the SIS assumption, the number of

additions in the supported circuits depends on how short are the vectors produced by MATRIXSAMPLEPRE and MATRIXSAMPLEDOM.

- **Addition with constant:** Given a message m_1 and a constant c, if we have $\mathcal{C}(m_1) = m_1 + c \mod q$, we can produce $\mathcal{C}_R$ and $\mathcal{C}_D$ in the following way:

$$\mathcal{C}_R(\mathbf{R}_1) = \mathbf{R}_1 \qquad \mathcal{C}_D(\mathbf{D}_1) = \mathbf{D}_1 + c\mathbf{G}$$

This works because if $CH_{HTDF}.\text{HASH}(ek, m_1, \mathbf{R}_1) = \mathbf{D}_1$, then by the definition of CH_{HTDF}:

$$\begin{aligned} CH_{HTDF}.\text{HASH}(ek, m_1 + c, \mathbf{R}_1) &= (m_1 + c)\mathbf{G} + \mathbf{A}\mathbf{R}_1 \\ &= c\mathbf{G} + m_1\mathbf{G} + \mathbf{A}\mathbf{R}_1 \\ &= c\mathbf{G} + CH_{HTDF}.\text{HASH}(ek, m_1, \mathbf{R}_1) \end{aligned}$$

If $||\mathbf{R}_1||_\infty < \beta'$, then the resulting signature after this operation remains with the same size β'. There is no limit in the number of additions with constant values that we can have in circuits in set C.

- **Multiplication of messages:** Given two messages $m_1, m_2 \in M$, we can define $\mathcal{C}(m_1, m_2) = m_1 m_2 \mod q$. Given this algorithm, we can define $\mathcal{C}_R$ and $\mathcal{C}_D$ first computing a matrix $\mathbf{X}$ with a short norm such that $\mathbf{G}\mathbf{X} = CH_{HTDF}.\text{HASH}(ek, m_1, \mathbf{R}_1)$. Then, we define the two derived algorithms as:

$$\mathcal{C}_R(\mathbf{R}_1, \mathbf{R}_2) = m_2\mathbf{R}_1 + \mathbf{R}_2 X \mod q \qquad \mathcal{C}_D(\mathbf{D}_1, \mathbf{D}_2) = \mathbf{D}_2\mathbf{X} \mod q$$

The reason becausse it works is that if $CH_{HTDF}.\text{HASH}(ek, m_1, \mathbf{R}_1) = \mathbf{D}_1$ and $CH_{HTDF}.\text{HASH}(ek, m_2, \mathbf{R}_2) = \mathbf{D}_2$, then:

$$
\begin{aligned}
CH_{HTDF}.\text{HASH}(ek, m_1 m_2, m_2\mathbf{R}_1 + \mathbf{R}_2\mathbf{X}) &= m_1 m_2\mathbf{G} + \mathbf{A}(m_2\mathbf{R}_1 + \mathbf{R}_2\mathbf{X}) \\
&= m_1 m_2\mathbf{G} + m_2\mathbf{A}\mathbf{R}_1 + \mathbf{A}\mathbf{R}_2\mathbf{X} \\
&= m_2(m_1\mathbf{G} + \mathbf{A}\mathbf{R}_1) + \mathbf{A}\mathbf{R}_2\mathbf{X} \\
&= m_2\mathbf{D}_1 + \mathbf{A}\mathbf{R}_2\mathbf{X} \\
&= m_2\mathbf{D}_1 + (\mathbf{A}\mathbf{R}_2\mathbf{X} + m_2\mathbf{D}_1\mathbf{X} - m_2\mathbf{D}_1\mathbf{X}) \\
&= m_2\mathbf{D}_1 + (\mathbf{A}\mathbf{R}_2\mathbf{X} + \mathbf{D}_2\mathbf{X} - \mathbf{D}_2\mathbf{X}) \\
&= m_2\mathbf{D}_1 + (\mathbf{A}\mathbf{R}_2 X + \mathbf{D}_2\mathbf{X} - (m_2\mathbf{G}\mathbf{X} + \mathbf{A}\mathbf{R}_2 X)) \\
&= m_2\mathbf{D}_1 + (\mathbf{A}\mathbf{R}_2\mathbf{X} + \mathbf{D}_2\mathbf{X} - (m_2\mathbf{D}_1 + \mathbf{A}\mathbf{R}_2 X)) \\
&= \mathbf{D}_2\mathbf{X}
\end{aligned}
$$

In this case, if $||\mathbf{R}_1||_\infty < \beta'$ and if $||\mathbf{R}_2||_\infty < \beta'$, then $||m_2\mathbf{R}_1 + \mathbf{R}_2 X||_\infty < (|m_2| + m)\beta$. This means that we can support multiplication between messages in M only if one of the multiplicands is a sufficiently small value. Otherwise, the norm of the randomness for CH_{HTDF} that we use as a signature would be too big to be considered valid and secure using the SIS assumption.

- **Multiplication by constant:** Given a message m_1 and a constant c, if we have $\mathcal{C}(m_1) = c(m_1) \mod q$, we can produce $\mathcal{C}_R$ and $\mathcal{C}_D$ first computing a matrix $\mathbf{X} \in \mathbb{Z}_q^{m \times m}$ such that $\mathbf{G}\mathbf{X} = a\mathbf{G}$. After this, we can compute the two derived algorithms as:

$$\mathcal{C}_R(\mathbf{R}_1) = \mathbf{R}_1 X \qquad \mathcal{C}_D(\mathbf{D}_1) = \mathbf{D}_1\mathbf{X}$$

This works because if $CH_{HTDF}.\text{HASH}(ek, m_1, \mathbf{R}_1) = \mathbf{D}_1$, then:

$$
\begin{aligned}
CH_{HTDF}.\text{HASH}(ek, c(m_1), \mathbf{R}_1\mathbf{X}) &= c(m_1)\mathbf{G} + \mathbf{A}\mathbf{R}_1\mathbf{X} \\
&= c(m_1)\mathbf{G} + \mathbf{A}\mathbf{R}_1\mathbf{X} + \mathbf{D}_1\mathbf{X} - \mathbf{D}_1\mathbf{X} \\
&= c(m_1)\mathbf{G} + \mathbf{A}\mathbf{R}_1\mathbf{X} + \mathbf{D}_1\mathbf{X} - (m_1\mathbf{G} + \mathbf{A}\mathbf{R}_1)\mathbf{X} \\
&= c(m_1)\mathbf{G} + \mathbf{A}\mathbf{R}_1\mathbf{X} + \mathbf{D}_1\mathbf{X} - (m_1\mathbf{G}\mathbf{X} + \mathbf{A}\mathbf{R}_1\mathbf{X}) \\
&= c(m_1)\mathbf{G} + \mathbf{A}\mathbf{R}_1\mathbf{X} + \mathbf{D}_1\mathbf{X} - (c(m_1)\mathbf{G} + \mathbf{A}\mathbf{R}_1\mathbf{X}) \\
&= \mathbf{D}_1\mathbf{X}
\end{aligned}
$$

Here, if $||\mathbf{R}_1||_\infty < \beta'$, then $||\mathbf{R}_1\mathbf{X}||_\infty < m\beta'$.

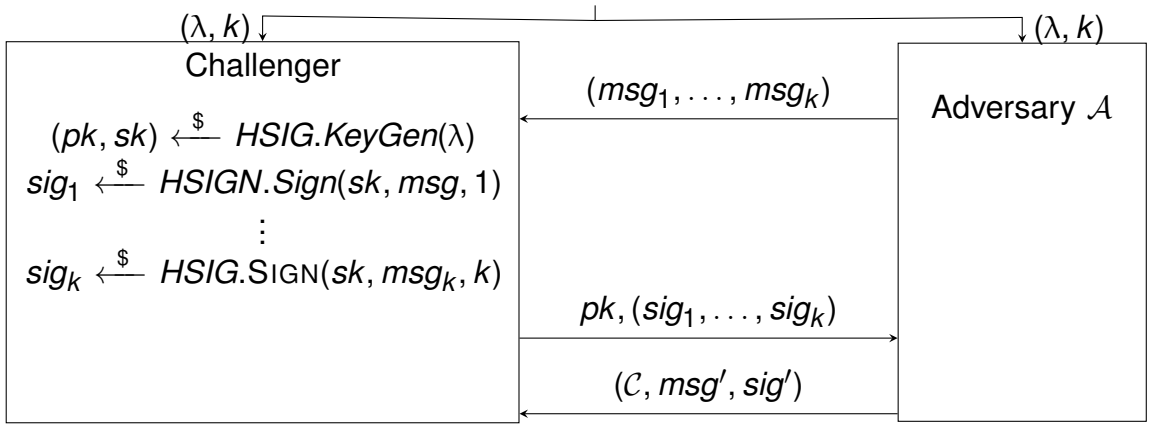

Figure 27 – Attack Game: One-Time Signature Security Against Weak Chosen Message Attack for Homomorphic Signatures

Combining recursively the four kind of operations above we can build any kind of algorithm that can be expressed using only addition and multiplication. The number of operations that we can combine depends on how short are the euclidean norm of our signatures.

Now we will define the security of a one-time homomorphic signature using the following attack game:

Attack Game 17 ***(Weak unforgeability for one-time homomorphic signature against chosen message attack).*** *For a given homomorphic signature scheme HSIG defined over (M, S, C), the adversary $\mathcal{A}$ and the challenger are initialized by the security parameter λ and a positive integer k.*

The adversary first chooses a list with k target messages $(msg_1, \ldots, msg_k) \in M^k$ and send to the challenger. The challenger runs $(pk, sk) \xleftarrow{\$} HSIG.\text{KEYGEN}(\lambda, k)$, signs each message msg_j running $HSIG.\text{KEYGEN}(sk, msg_j, j)$ producing a signature sig_j for $1 \leq j \leq k$.

The challenger sends to the adversary the public key pk and the signature tuple $(sig_1, \ldots, sig_k)$. After this, the challenger outputs some circuit $\mathcal{C} \in C$, a message $msg' \in M$ and a signature $sig' \in S$.

We say that the adversary wins the game if $HSIG.\text{VERIFY}(pk, \mathcal{C}, msg', sig') = \texttt{accept}$ and $\mathcal{C}(msg_1, \ldots, msg_k) \neq msg'$.

We denote by $HCMAadv_{1\text{-}Weak}[\mathcal{A}, HSIG]$ the probability that a given adversary $\mathcal{A}$ wins this game for a given homomorphic signature scheme HSIG.

With this security definition we can prove that our homomorphic signature *HSIG* is secure.

Theorem 27 *If the chameleon hash CH_{HTDF} is collision-resistant, then the homomorphic signature HSIG defined is a secure one-time signature against weak chosen message attack.*

Proof: Assuming that we have an adversary $\mathcal{A}$ that forges a homomorphic signature in the above attack game, we will build an adversary $\mathcal{B}$ that finds collisions for the chameleon hash CH_{HTDF}. Our adversary $\mathcal{B}$ works as below:

- **Initialization:** Adversary $\mathcal{B}$ gets from its challenger a key *ek* for the chameleon hash CH_{HTDF}. Then it initializes $\mathcal{A}$ with the security parameter λ and any arbitrary k. Adversary $\mathcal{A}$ send a list of k messages $(m_1, \ldots, m_k)$.

  Now $\mathcal{B}$ chooses k random matrices $(\mathbf{R}_1, \ldots, \mathbf{R}_k)$ using the previously seen algorithm MATRIXSAMPLEDOM. With these values, it computes $(\mathbf{D}_1, \ldots, \mathbf{D}_k)$ as the resulting digests for:

  $$CH_{HTDF}.\text{HASH}(ek, m_1, \mathbf{R}_1), \ldots, CH_{HTDF}.\text{HASH}(ek, m_k, \mathbf{R}_k)$$

  . Finally, it sends $(ek, (\mathbf{D}_1, \ldots, \mathbf{D}_k))$ to $\mathcal{A}$ as the homomorphic signature public key and also sends $(\mathbf{R}_1, \ldots, \mathbf{R}_k)$ as the signature for the messages.

- **Finalization:** Adversary $\mathcal{A}$ outputs $(\mathcal{C}', m', \mathbf{R'})$. If this is an accepted signature, then CH_{HTDF}.EK, M', R' = $\mathcal{C}_D(\mathbf{D}_1, \ldots, \mathbf{D}_k)$.

  If the adversary $\mathcal{A}$ wins, then $m' \neq \mathcal{C}(m_1, \ldots, m_k)$. This means that $\mathcal{B}$ can produce a collision in the chameleon hash using the pairs $(\mathcal{C}(m_1, \ldots, m_k), \mathcal{C}_R(\mathbf{R}_1, \ldots, \mathbf{R}_k))$ and (m', R').

The adversary $\mathcal{B}$ always succeed when $\mathcal{A}$ succeed. Therefore, we have that for all adversaries $\mathcal{A}$, we can build adversary $\mathcal{B}$ such that:

$$HCMAadv_{1\text{-}Weak}[\mathcal{A}, HSIG] \leq CRadv[\mathcal{B}, CH_{HTDF}]$$

By assumption, our chameleon hash is collision-resistant, therefore our signature is secure when used a single time against a weak chosen-message attack. ■

One of the problems of this basic construction is the weak security definition. However, this can be improved with some modifications. In (GORBUNOV et al., 2015) one suggestion is improving the signature with the same technique presented in from Subsection 3.4.3 or in Subsection 4.3.1 to increase the security in a signature scheme. The same methods that transform a signature secure against generic chosen message attacks or random message attack in one secure against adaptive chosen message attacks can also be employed here to transform a signature with a weak security in a one-time signature in a standard security definition, where the adversary knows the public key when it sends the signing query.

However, as we are using an homomorphic signature, the chameleon hash used to increase the signature security also must be an homomorphic chameleon hash. If we are already using a homomorphic chameleon hash to create the signature *HSIG*,

then this means using the chameleon hash twice. The first chameleon hash must have as digest space a subset of the message space of *HSIG*.

Assuming that we already have the one-time homomorphic signature *HSIG* with a weak security, we can create an one-time homomorphic signature using the standard security definition using the homomorphic chameleon hash *CH* as below:

- $HSIG'.\textsc{KeyGen}(\lambda, k)$:
  1. $(ek, tk) \xleftarrow{\$} CH.\textsc{KeyGen}(\lambda)$
  2. $(pk, sk) \xleftarrow{\$} HSIG.\textsc{KeyGen}(\lambda, k)$
  3. $pk' \leftarrow (ek, pk)$
  4. $sk' \leftarrow (ek, sk)$
  5. **return** (pk', sk')
- $HSIG'.\textsc{Sign}(sk', msg, i)$:
  1. $(ek, sk) \leftarrow sk'$
  2. $rnd \xleftarrow{\$} R$
  3. $dgt \leftarrow CH.\textsc{Hash}(ek, msg, rnd)$
  4. $sig \xleftarrow{\$} HSIG.\textsc{Sign}(sk, dgt, i)$
  5. $sig' \leftarrow (dgt, rnd, sig)$
  6. **return** sig'
- $HSIG'.\textsc{Eval}(pk', \mathcal{C}, (msg_1, sig'_1), \ldots, (msg_k, sig'_k))$:
  1. $(ek, pk) \leftarrow pk'$
  2. From $\mathcal{C}$, derive $\mathcal{C}_R$ and $\mathcal{C}_D$ for *CH*
  3. **for** i **in** $\{1, \ldots, k\}$:
  4. $\quad (dgt_i, rnd_i, sig_i) \leftarrow sig'_i$
  5. $dgt' \leftarrow \mathcal{C}_D(dgt_1, \ldots, dgt_k)$
  6. $rnd' \leftarrow \mathcal{C}_R(rnd_1, \ldots, rnd_k)$
  7. $sig' \leftarrow HSIG.\textsc{Eval}(pk, (dgt_1, sig_1), \ldots, (dgt_k, sig_k))$
  8. **return** (dgt', rnd', sig')
- $HSIG'.\textsc{Verify}(pk', \mathcal{C}, msg, (dgt, rnd, sig))$:
  1. $(ek, pk) \leftarrow pk'$
  2. **if** $CH.\textsc{Hash}(ek, msg, rnd) \neq dgt$:

3. **return** `reject`
4. **return** $HSIG.\text{VERIFY}(pk, dgt, sig)$

One problem for our construction using CH_{HTDF} is that using the chameleon hash twice, the norm of our signature could increase more, depending of how is defined the chameleon hash CH. In this case, the cost of implementing this more rigorous security model is a decrease in the number of possible circuits compatible with the signature.

Theorem 28 *If the homomorphic signature HSIG defined over* (M, S, C) *is secure as a one-time signature against chosen message attack in the weak security model and CH is a collision-resistant homomorphic chameleon hash with preimage uniformity defined over* (M', R, D) *with* $D \subseteq M$*, then the signature* $HSIG'$ *is secure as a one-time signature against a chosen message attack in the standard security model.*

Proof: Let's assume that we have an adversary $\mathcal{A}$ that breaks the security of $HSIG'$ with a single chosen message attack in the standard security. We will use it to build an adversary $\mathcal{B}$ that breaks the security of $HSIG$ in a chosen message attack in the weak security.

Adversary $\mathcal{B}$ works as below:

- **Initialization**: Adversary $\mathcal{B}$ is initialized with (λ, k). First it chooses a tuple of digests $(dgt_1, \ldots, dgt_k)$ with elements chosen uniformly at random from D and sends to its challenger. It gets as response a public key pk and a list of signatures $(sig_1, \ldots, sig_k)$ for each sent digest. It also produces a pair of keys (ek, tk) for the homomorphic chameleon hash used in the scheme and initializes adversary $\mathcal{A}$ sending the public key (ek, pk).
- **Query:** Knowing the public key, adversary $\mathcal{A}$ sends a query $(msg_1, \ldots, msg_k)$. For each message msg_i in this tuple, adversary $\mathcal{B}$ produces a corresponding rnd_i running $CH.\text{PREIMAGE}(tk, msg_i, dgt_i)$. Finally, $\mathcal{B}$ sends as response the signature for each message msg_i as (dgt_i, rnd_i, sig_i). If the chameleon hash has the preimage uniformity property, then each produced rnd_i has the same probability distribution as a randomness chosen uniformly at random as required.
- **Finalization:** After the single query, adversary $\mathcal{A}$ produces a possible forgery $(\mathcal{C}, msg', (dgt', rnd', sig'))$. If the forgery is accepted, then $\mathcal{C}(msg_1, \ldots, msk_k) \neq msg'$. If $\mathcal{C}_D(dgt_1, \ldots, dgt_k) = dgt'$, this means that we found a collision in the chameleon hash. If not, then adversary $\mathcal{B}$ outputs as forgery the tuple $(\mathcal{C}, dgt', sig')$.

The lower bound for the success of $\mathcal{B}$ is given by the probability of $\mathcal{A}$ winning its attack game minus the probability that it wins finding a collision in CH. The probability

of finding a collision in *CH* can be modelled as $CRadv[\mathcal{B}'CH]$ for some adversary $\mathcal{B}'$. Therefore, the lower bound is:

$$HCMAadv_{1\text{-}Weak}[\mathcal{B}, HSIG] \geq HCMAadv_1[\mathcal{A}, HSIG'] - CRadv[\mathcal{B}', CH]$$

Rewriting this as an upper bound for adversary $\mathcal{A}$, we conclude that for all adversaries $\mathcal{A}$, we can build adversaries $\mathcal{B}$ and $\mathbf{B}'$ such that:

$$HCMAadv_1[\mathcal{A}, HSIG'] \leq HCMAadv_{1\text{-}Weak}[\mathcal{B}, HSIG] + CRadv[\mathcal{B}', CH]$$

As by our assumptions the right side of this inequation is negligible, this means that no adversary can forge signatures for the homomorphic one-time signature *HSIG*′, except with negligible probability. ■

To make possible using this signature more than once without the need of regenerating new keys, there are some alternatives proposed like combining this construction with labelled signatures or labelled chameleon hash functions. These techniques are proposed both in (GORBUNOV et al., 2015) and in (XIE et al., 2017). However, such techniques are outside the scope of this work.

6 POST-QUANTUM SIGNATURE WITH PREIMAGE CHAMELEON HASHING

In this chapter we present our novel signature scheme, that unlike the previous signatures described in the last chapter, achieves unforgeability under adaptive chosen message attacks against both classical and post-quantum adversaries. We let the adversary performs a polynomially bounded number of queries. The construction is secure provided that the Ring-SIS assumption is true (Subsection 2.2.4).

Other than this, the construction also has an additional interesting property. It allows the signer to encode in its signature custom and non-random information that can be checked during the signature verification.

This chapter is organized as follows:

- In section 6.1, we define preimage signatures, a signature scheme where the signer can store custom and non-random data in pairs composed of a message and a valid signature.
- In section 6.2, we provide a construction for our preimage signature using a suitable chameleon hash. We also show its security proof.
- In section 6.3, we present results about the implementation of the scheme.

6.1 PREIMAGE SIGNATURES

We define a preimage signature *PSIG* as a signature scheme defined over sets (M, S, D) where M is the message space, S is the signature space and D is the data space. The scheme is composed of four algorithms (*PSIG*.KEYGEN, *PSIG*.SIGN, *PSIG*.VERIFY, *PSIG*.EXTRACT). In a preimage signature we have the following properties:

- *PSIG*.KEYGEN is a probabilistic algorithm that takes as input (λ, dt) where λ is the security parameter and $dt \in D$ is any arbitrary data. It outputs a pair of keys (pk, sk) like in a regular signature.
- *PSIG*.SIGN is a probabilistic algorithm that takes as input (sk, msg) where sk is the secret key and $msg \in M$. It outputs a signature $sig \in S$.
- *PSIG*.VERIFY is a deterministic algorithm that takes as input (pk, msg, sig) where pk is a public key, $msg \in M$, $sig \in S$. It outputs `accept` or `reject`. It outputs `accept` only if the signature sig was generated for the corresponding message msg.
- *PSIG*.EXTRACT is a deterministic algorithm that takes as input (pk, msg, sig) like the *PSIG*.VERIFY algorithm. If the signature sig was generated for the message

msg using the corresponding secret key *sk*, it outputs $dt \in D$ chosen during key generation.

The security model for this kind of signature must take in consideration that the data $dt \in D$ chosen during key generation is non-random and probably has some underlying meaning. To model this, we will define the security of this signature using the usual adaptive chosen message attack game, but we will let the adversary choose any desired *dt* before the key generation. Our attack game is given below:

Attack Game 18 ***(Unforgeability against adaptive chosen message attacks for preimage signatures).*** *For a preimage signature scheme PSIG defined over* (M, S, D)*, both an adversary* $\mathcal{A}$ *and the challenger are initialized by a security parameter* λ*.*

Next, the adversary $\mathcal{A}$ *choose some* $dt \in D$ *and sends to the challenger.*

The challenger produces a pair of keys (pk, sk) *with PSIG.*KEYGEN(λ, dt) *and sends pk to the adversary.*

The adversary can make a polynomially bounded number of adaptive signing queries sending in each of them a message msg_i *to the challenger. The challenger computes a signature* sig_i *computing PSIG.*SIGN(sk, msg_i) *and sends it to the adversary.*

In the end the adversary outputs a forgery (msg', sig')*. We say that the adversary wins the game if* $(msg', sig') \neq (msg_i, sig_i)$ *for all pairs* (msg_i, sig_i) *produced during signing queries and if SIG.*VERIFY$(pk, msg', sig') = \texttt{accept}$ *or if SIG.*EXTRACT$(pk, msg', sig') = dt$*.*

We denote by PCMAadv$[\mathcal{A}, PSIG]$ *the probability of some adversary* $\mathcal{A}$ *winning this attack game against a challenger using the preimage signature PSIG. We say that a preimage signature PSIG is unforgeable against adaptive chosen message attacks if for all efficient adversaries* $\mathcal{A}$*, the value of PCMAadv*$[\mathcal{A}, PSIG]$ *is negligible.*

Such signature scheme can have the following applications:

- More user-friendly software for signature verification: a signer could store in its signatures some *dt* that represents an image of a handwritten signature of the signer picture. Such image can be presented in the screen if the signature is correctly verified.
- Linking digital signatures with biometric data: the data *dt* encoded in a user signature could represent some biometric data about him. For example, its fingerprint.
- Alternative signature verification: Instead of using the *PSIG.*VERIFY algorithm, in some contexts, the *PSIG.*EXTRACT algorithm could be used to check the validity of a signature. The verifier could extract using EXTRACT$(pk, msg, sign)$ some fixed

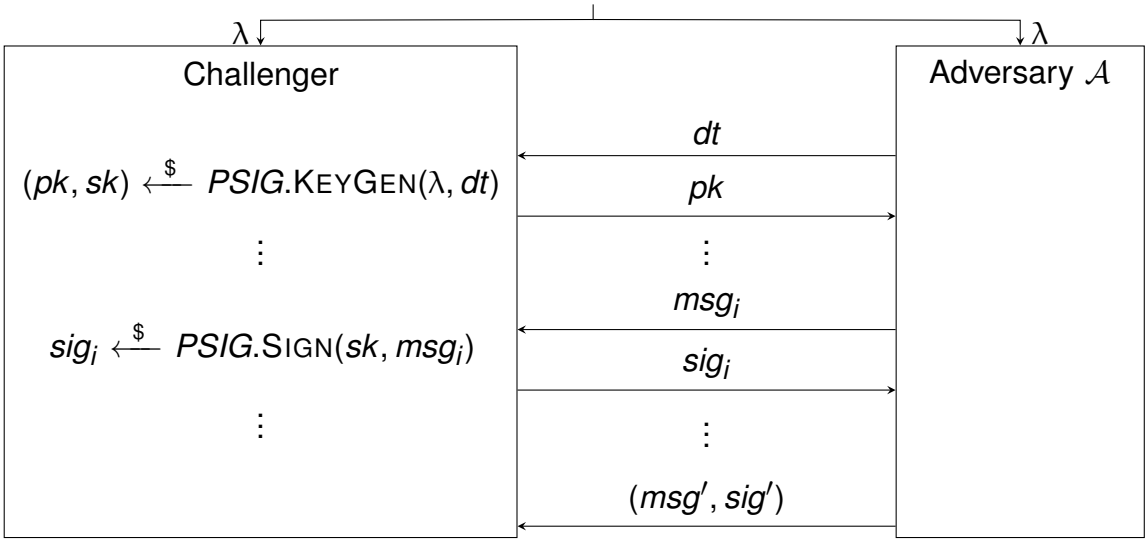

Figure 28 – Attack Game: Unforgeability against adaptive chosen message attack for preimage signatures.

information about the signer, which, if valid and correct, attests the validity of the signature.

6.2 CONSTRUCTION USING PREIMAGE CHAMELEON HASH FUNCTIONS

We can build a preimage signature using a construction very similar to the one-time signature scheme presented in Section 5.1. We just replace the regular chameleon hash with a preimage chameleon hash to let the user choose the encoded data during key generation. To create a preimage signature defined over sets (M, S, D), we use a preimage chameleon hash defined over the same sets (M, S, D):

- $PSIG.\text{KEYGEN}(\lambda, dt)$:
  1. $(ek, tk) \xleftarrow{\$} CH.\text{KEYGEN}(\lambda)$
  2. $pk \leftarrow (ek, dt)$
  3. $sk \leftarrow (tk, dt)$
  4. **return** (pk, sk)
- $PSIG.\text{SIGN}(sk, msg)$:
  1. $(tk, dt) \leftarrow sk$
  2. **return** $CH.\text{PREIMAGE}(tk, msg, dt)$
- $PSIG.\text{EXTRACT}(pk, msg, sig)$:
  1. $(ek, dt) \leftarrow pk$
  2. **return** $CH.\text{HASH}(ek, msg, sig)$

- SIG_+.VERIFY(pk, msg, sig):
  1. $(ek, dt) \leftarrow pk$
  2. **if** CH.HASH(ek, msg, sig) = dt:
  3. **return** `accept`
  4. **else**:
  5. **return** `reject`

However, we cannot use any generic chameleon hash to build a secure preimage signature. As seen in Section 5.1, with a generic chameleon hash we could prove the signature unforgeability using only a very weak security model. The main problem is that most versions of chameleon hash not necessarily remain collision-resistant if some sample collisions are revealed. As in this construction we can reveal new collisions in CH each time a new signature is revealed, there are chameleon hash functions for which this construction becomes completely insecure.

To produce a preimage signature unforgeable against adaptive chosen message attacks, that also can have its keys used more than once and is secure in a post-quantum model, we will propose a specific construction.

Our chameleon hash will be based on the CH_{SIS} chameleon hash defined in Subsection 4.2.2.4. The greatest difference is that we will use the Ring-SIS assumption instead of the SIS assumption. The construction will then produce smaller keys and we will be able to stress better its similarities with the signature SIG_{GPV} defined in Section 5.4.2.

We will denote this chameleon hash construction by CH_{RSIS} and it is defined over sets (M, R', D) where R' is the subset of elements from the modular polynomial ring R with short euclidean norm (for using in the preimage signature we can assume that their euclidean norm is smaller than $\beta/2$) and D is the modular polynomial ring R_q whose coefficients are always modulo q. The construction also uses a collision-resistant hash function $HASH : M \rightarrow D$.

We define CH_{RSIS} algorithms as:

- CH_{RSIS}.KEYGEN(λ):
  1. $(R_q, m, q, \beta) \leftarrow$ RSISPARAMS(λ)
  2. $(\mathbf{a}, \mathbf{T}) \xleftarrow{\$}$ RINGTRAPGEN($R_q, m, n, q, \mathcal{D}$)
  3. $ek \leftarrow (R_q, m, q, \beta, \mathbf{a})$
  4. $tk \leftarrow (R_q, m, q, \beta, \mathbf{a}, \mathbf{T})$
  5. **return** (ek, tk)
- CH_{RSIS}.HASH(ek, msg, $\mathbf{r}$):

1. $(R_q, m, q, \beta, \mathbf{a}) \leftarrow ek$
2. **return** $HASH(msg) + \mathbf{ar}$

- CH_{RSIS}.PREIMAGE$(tk, msg, y())$:

1. $(R_q, m, q, \beta, \mathbf{a}, \mathbf{T}) \leftarrow tk$
2. **return** RINGSAMPLEPRE$_{\beta/2}(R_q, m, q, \mathbf{a}, \mathbf{T}, y() - HASH(msg))$

In the proof for our preimage signature, we will model our hash function *HASH* as a random oracle.

We also will make another requirement about the use of this chameleon hash in the context of preimage signatures: the algorithm *CH*.PREIMAGE should always return the same value if given the same input. This can be achieved, for example, deriving the probabilistic decisions in RINGSAMPLEPRE from bits produced by a pseudo-random function fed with a fixed key chosen during key generation and with RINGSAMPLEPRE input.

Now we will prove the security of our preimage signature *PSIG* assuming that is is built using chameleon hash CH_{RSIS}:

Theorem 29 *If the Ring-SIS assumption is true and if the algorithm* RINGTRAPGEN *(defined in Subsection 5.4.1) produces vectors indistinguishable from random and uniform, then the signature scheme PSIG defined above using the chameleon hash CH_{RSIS} is secure against adaptive chosen message attacks in the random oracle model.*

Proof: This proof is a generalization of the proof for the SIG_{GPV} signature presented at Subsection 5.4.2.

We will show that if we have an adversary $\mathcal{A}$ that can forge a signature for *PSIG*, we can build an adversary $\mathcal{B}$ that wins the attack game for the Ring-SIS assumption.

Our adversary $\mathcal{B}$ is defined as:

- **Initialization:** Adversary $\mathcal{B}$ takes as input the random and uniform vector $\mathbf{a}$ and the Ring-SIS public parameters (R_q, m, q, β). Next, it initializes adversary $\mathcal{A}$ with the security parameter. Adversary $\mathcal{A}$ outputs a data to be encoded in the signature $d() \in R_q$. The adversary $\mathcal{B}$ produces the public key using the chameleon hash key $ek = (R_q, m, q, \beta, \mathbf{a})$ and the polynomial $d()$. It sends $pk = (ek, dt)$ to adversary $\mathcal{A}$.
- **Random Oracle Query:** For each msg_i queried, if it is a message never queried before, we will first choose randomly a signature $\mathbf{r}_i$ using RINGSAMPLEDOM. Knowing its signature, we deduce its correct random oracle output computing $d() - \mathbf{a} \cdot \mathbf{r}_i$. Notice that by definition of algorithm RINGSAMPLEDOM, as $\mathbf{a}$ is random and uniform, the result of $\mathbf{a} \cdot \mathbf{r}_i$ is also indistinguishable from a value chosen uniformly at random. Therefore, this computed result is a valid response for a random

oracle query. We memorize this random oracle response and the signature for message msg_i. If this message is queried again we will send the same response.

- **Signing Query:** When adversary $\mathcal{A}$ send a signing query msg_i, adversary $\mathcal{B}$ checks if this message was queried before in a random oracle query or signing query. If so, it already has a memorized signature $\mathbf{r}_i$ associated with it. Adversary $\mathcal{B}$ sends this $\mathbf{r}_i$ as response.

  If msg_i never was queried before, adversary $\mathcal{B}$ produces a new random signature $\mathbf{r}_i$ running RINGSAMPLEDOM, memorizes that this signature is associated with msg_i and sends $\mathbf{r}_i$ as response.

- **Finalization:** In the end adversary $\mathcal{A}$ produces a forgery (msg', **r'**).

  If msg' never was used in a random oracle or signing query, adversary $\mathcal{B}$ simulates a random oracle query for msg' as described above. Doing so, now we have a signature **r''** associated with msg' that was produced by this random oracle query simulation. If msg' had already been sent in a random oracle or signing query, this is not necessary as we already have a signature **r''** associated with msg' that was produced by a query.

  This means that:

$$HASH(msg') + \mathbf{ar'} = HASH(msg') + \mathbf{ar''} = d()$$

  And:

$$\mathbf{a}(\mathbf{r'} - \mathbf{r''}) = 0$$

  This means that if **r'** $\neq$ **r''**, then $\mathcal{B}$ can produce a correct output for the Ring-SIS attack game with (**r'** $-$ **r''**).

Like in the proof for the SIG_{GPV} signature scheme, we can assume that each message can have many different signatures and no signature has a probability greater than 1/2 of being chosen by our algorithm RINGSAMPLEPRE. Therefore, we have a probability of at least 1/2 that **r'** $\neq$ **r''**.

The probability of $\mathcal{A}$ producing a correct forgery in this scenario is given by $PCMAadv^{RO}[\mathcal{A}, PSIG]$ minus the probability that it does so exploring some weakness in the algorithm RINGTRAPGEN (which as usual we model as the probability $WPRFadv[\mathcal{B}', PRF_{\text{RINGTRAPGEN}}]$).

Our lower bound for the probability that $\mathcal{B}$ finds a correct solution in the Ring-SIS attack game is given by:

$$RSISadv[\mathcal{B}] \geq \frac{1}{2} PCMAadv^{RO}[\mathcal{A}, PSIG] - WPRFadv[\mathcal{B}', PRF_{\text{TRAPGEN}}]$$

Rewriting this inequation, we conclude that for all adversaries $\mathcal{A}$ that can create a forgery for our signature SIG_+, we can create adversaries $\mathcal{B}$ and $\mathcal{B}'$ such that:

$$PCMAadv^{RO}[\mathcal{A}, PSIG] \leq 2(RSISadv[\mathcal{B}] + WPRFadv[\mathcal{B}', PRF_{\text{RINGTRAPGEN}}])$$

By our assumptions, the Ring-SIS is a hard problem and RINGTRAPGEN produces vectors **a** computationally indistinguishable than random and uniform vectors. Therefore, no adversary can create forgeries in our signature, except with negligible probability.

Finally, notice that both **r'** and **r"** obtained by our adversary $\mathcal{B}$ are values with euclidean norm smaller than $\beta/2$. Therefore, the output (**r'**−**r"**) produced by $\mathcal{B}$ has norm smaller than β as required by the Ring-SIS assumption. ■

Like in the proof for the GPV signature, here adversary $\mathcal{B}$ treated each query homogeneously. Except when checking if a query was already sent in the past to recover the same response, adversary $\mathcal{B}$ ignores the historic of previous events when computing its responses, do not treat any query differently and do not use values sent in a query to deduce something about the incoming forgery. It also does not execute adversary $\mathcal{A}$ more than once expecting the same outputs while feeding it with the same input.

This means that this security proof can be adapted to post-quantum scenarios. Our proposed signature is also secure against quantum adversaries.

6.3 IMPLEMENTATION, RESULTS AND DISCUSSION

To evaluate the performance of our proposed construction, we implemented the chameleon hash CH_{RSIS} in the form of a preimage signature *PSIG* using the library by (ROHLOFF et al., 2020) implementing the ring-based construction from (EL BANSARKHANI; BUCHMANN, 2014) using the same method to derive the key from the trapdoor. The polynomial ring was R_q with q having $k = 27$ bits. The same parameters were used both for the GPV signature and our construction. We choose these parameters to achieve about 100 bits of security according with suggestions from (EL BANSARKHANI; BUCHMANN, 2014), which used the framework from (RÜCKERT; SCHNEIDER, 2010).

To compare the execution time, we measured in the same machine the following signatures: RSA and ECDSA from OpenSSL 1.1.1 library, CRYSTALS-Dilithium (proposed in (DUCAS, Léo et al., 2018)) and FALCON (implementation proposed in (PORNIN, 2019)) from code submitted the NIST standardization project and a BLISS-B (described in (DUCAS, Léo, 2014)) implementation from strongSwan library version 5.8.4 (STRONGSWAN, 2020). The ECDSA used the curve B-233.

All tests were run in a computer with a Dual-Core Intel Pentium B980 2.40GHz (without AVX2 support) with 4GB of memory and running Ubuntu 18.04.4. While mea-

suring running time, the tests were performed 1000 times, and the mean was extracted. Given the measured standard deviation, we computed the error margin given an interval of confidence of 95%, assuming a normal distribution. For all schemes, the signature time also includes the time to perform a hash on the messages to be signed. All of them use SHA256, except for CRYSTALS-Dilithium and FALCON, which used SHAKE256. The code used to measure running times and sizes can be checked in (ASTRIZI, 2020).

Table 1 – Running time comparison [ms]. The confidence interval is 95%.

Scheme	Security level	KeyGen	Sign/Preimage	Verify/Hash
RSA 2048	112	160.541 ± 6.540	2.620 ± 0.001	0.053 ± 0.000
ECDSA 233	112	0.661 ± 0.009	0.689 ± 0.001	1.338 ± 0.002
Dilithium 1280×1024	128	0.475 ± 0.023	1.719 ± 0.076	0.453 ± 0.000
BLISS-B I	128	916.577 ±14.900	2.448 ± 0.221	0.224 ± 0.020
FALCON-512	≈100	15.088 ± 0.542	0.629 ± 0.018	0.079 ± 0.000
GPV (n=512, k=27)	≈100	5.521 ± 0.026	32.005 ± 0.033	0.241 ± 0.008
CH_{RSIS} ($n = 512, k = 27$)	≈100	5.520 ± 0.021	32.232 ± 0.070	0.244 ± 0.008

Table 2 – Size comparison [bytes].

Scheme	Digest	σ	*pk*	*sk*
RSA 2048	-	256	259	512
ECDSA 233	-	58	31	29
Dilithium 1280×1,024	-	2,829	1,472	3,504
BLISS-B I	-	732	933	1182
FALCON-512	-	651	897	1281
GPV (n = 512, k = 27)	-	61,440	61,440	114,688
CH_{RSIS} (n = 512, k = 27)	2048	61,440	61,440	114,688

The results of the running times are summarized in Table 1. The sizes for signatures, keys, and chameleon hash digests are summarized in Table 2. As expected, our construction has comparable performance and the same key size as a GPV signature.

The tests and implementation prove the viability of the construction. Despite slower and with bigger keys than classical and more modern post-quantum signature schemes, our presented construction is, at the present moment, the only one known to implement post-quantum digital signature securely based on preimage chameleon hash. The bigger keys and slower signature usually is not a problem when using the signature scheme in modern computers, but can be a limiting factor when using this in embedded devices, where there is less storage and the battery consumption is a more critical concern. Finding new preimage chameleon hash schemes with performance on par with modern signature schemes is an open research problem.

7 CONCLUSION

In this dissertation, we analyzed some constructions of chameleon hash functions, their properties and especially previous applications of chameleon hash functions to build signature schemes. We ended the dissertation with a new proposal of signature construction using chameleon hash functions. The new construction, which we call a preimage signature, allows the signer to encode in its signature chosen information to be revealed during verification, a property that is novel to the best of our knowledge. The signature security comes from the security properties present in the chameleon hash function.

A difficulty that was overcome is keeping the chameleon hash secure, even if collisions are revealed and publicized during its usage. Even if an attacker learns some collisions seeing new signed messages, she cannot use this knowledge to produce a new forgery in our signature scheme, except with negligible probability. Our security proof is in the random oracle model and based on the Ring-SIS hardness assumption.

To the best of our knowledge, this is the first signature scheme based on chameleon hash functions where at the same time we can reutilize the same keys indefinitely, we can allow the adversary to choose messages adaptively to be signed without compromising the security, and this is guaranteed by a security proof that also can be adapted to post-quantum adversaries.

As the resulting signature scheme can be seen as a generalization of the GPV signature, we implemented a prototype adapting code from a C++ library that supported that signature. We also compared the execution times to other signatures, both classical and post-quantum. The comparison shows that the new signature algorithm is viable; that is, it can be used in practice to sign electronic documents.

The more interesting property of our signature is that the signature is part of a hash preimage of any value chosen by the signer during key creation. This opens new possibilities, like creating signatures where a signed message is verified comparing its hash after concatenated with the signature to check if the result is a given value with some special interest, for example, a representation of a handwritten signature. Such an approach can improve the user experience in terms of trust in the signature, not only regarding cryptographic verification, but also visual verification by recipients.

7.1 FURTHER WORKS

In order to use this new preimage signature scheme, further works include finding more efficient constructions of preimage signatures. As shown in our table comparing execution times, our preimage signature was running slower than the more recent post-quantum signatures. The signature FALCON-512 also was based on the GPV signature, but it implements much more optimizations and runs orders of magnitute faster than

our construction. This shows that we still have lots of room for improvements, we can study how to apply optimizations and techniques from FALCON-512, like the usage of the class of NTRU lattices.

As cryptographic assumptions like the Ring-SIS or SIS assumption are less studied and less used for cryptographic purposes compared to more used assumptions like the discrete logarithm, it is desirable to have other constructions of chameleon hash functions based in other assumptions believed to be secure against post-quantum adversaries. Having these other constructions, we still could use preimage signatures even if some weakness is found in the current Ring-SIS assumption.

Finally, our security model and proposed scheme can be improved and generalized. It would be useful if we could find a construction and security proof where we can ensure the security even if the signer can encode not only one, but many different chosen values.

It should also be interesting if we can ensure the security letting the adversary not only choose the target encoded data, but also define a list of sufficiently similar data, letting the adversary win the game not only by producing a signature with the chosen data, but also by producing a signature that encodes a sufficiently similar data. This could be useful when linking the preimage signature with biometric data, where some variation can be expected.

REFERENCES

AJTAI, Miklós. Generating hard instances of lattice problems. In: PROCEEDINGS of the twenty-eigth annual ACM symposium on Theory of computing. [S.l.: s.n.], 1996. P. 99–108.

ALAMATI, Navid; MONTGOMERY, Hart; PATRANABIS, Sikhar; ROY, Arnab. Minicrypt primitives with algebraic structure and applications. In: SPRINGER. ANNUAL International Conference on the Theory and Applications of Cryptographic Techniques. [S.l.: s.n.], 2019. P. 55–82.

AMOS, Ryan; GEORGIOU, Marios; KIAYIAS, Aggelos; ZHANDRY, Mark. One-shot signatures and applications to hybrid quantum/classical authentication. In: PROCEEDINGS of the 52nd Annual ACM SIGACT Symposium on Theory of Computing. [S.l.: s.n.], 2020. P. 255–268.

ASTRIZI, Thiago L. **Repository with preimage chameleon hash test source code**. [S.l.: s.n.], Sept. 2020. `https://github.com/thiagoharry/test_preimage_chameleon_hash`.

ATENIESE, Giuseppe; CHOU, Daniel H; DE MEDEIROS, Breno; TSUDIK, Gene. Sanitizable signatures. In: SPRINGER. EUROPEAN Symposium on Research in Computer Security. [S.l.: s.n.], 2005. P. 159–177.

ATENIESE, Giuseppe; MEDEIROS, Breno de. On the key exposure problem in chameleon hashes. In: SPRINGER. INTERNATIONAL Conference on Security in Communication Networks. [S.l.: s.n.], 2004. P. 165–179.

BACH, Eric. Discrete logarithms and factoring. University of California at Berkeley, 1984.

BELLARE, Mihir; NEVEN, Greagory. Multi-signatures in the plain public-key model and a general forking lemma. In: PROCEEDINGS of the 13th ACM conference on Computer and communications security. [S.l.: s.n.], 2006. P. 390–399.

BELLARE, Mihir; RISTOV, Todor. A characterization of chameleon hash functions and new, efficient designs. **Journal of cryptology**, Springer, v. 27, n. 4, p. 799–823, 2014.

BELLARE, Mihir; ROGAWAY, Phillip. Random oracles are practical: A paradigm for designing efficient protocols. In: PROCEEDINGS of the 1st ACM Conference on Computer and Communications Security. [S.l.: s.n.], 1993. P. 62–73.

BONEH, Dan; DAGDELEN, Özgür; FISCHLIN, Marc; LEHMANN, Anja; SCHAFFNER, Christian; ZHANDRY, Mark. Random oracles in a quantum world. In: SPRINGER. INTERNATIONAL conference on the theory and application of cryptology and information security. [S.l.: s.n.], 2011. P. 41–69.

BONEH, Dan; SHOUP, Victor. **A graduate course in applied cryptography**. version 0.5. [S.l.: s.n.], 2020. `https://cryptobook.us`.

BOUDOT, F.; GAUDRY, P.; GUILLEVIC, A.; HENINGER, N.; THOMÉ, E.; ZIMMERMANN, P. **Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment**. [S.l.: s.n.], 2020. Cryptology ePrint Archive, Report 2020/697. `https://eprint.iacr.org/2020/697`.

BRAKERSKI, Zvika; KALAI, Yael Tauman. A Framework for Efficient Signatures, Ring Signatures and Identity Based Encryption in the Standard Model. **IACR Cryptol. ePrint Arch.**, v. 2010, p. 86, 2010.

BRZUSKA, Christina; FISCHLIN, Marc; FREUDENREICH, Tobias; LEHMANN, Anja; PAGE, Marcus; SCHELBERT, Jakob; SCHRÖDER, Dominique; VOLK, Florian. Security of sanitizable signatures revisited. In: SPRINGER. INTERNATIONAL Workshop on Public Key Cryptography. [S.l.: s.n.], 2009. P. 317–336.

CASH, David; HOFHEINZ, Dennis; KILTZ, Eike; PEIKERT, Chris. Bonsai trees, or how to delegate a lattice basis. In: SPRINGER. ANNUAL international conference on the theory and applications of cryptographic techniques. [S.l.: s.n.], 2010. P. 523–552.

DAMGÅRD, Ivan Bjerre. Collision free hash functions and public key signature schemes. In: SPRINGER. WORKSHOP on the Theory and Application of Cryptograpjic Techniques. [S.l.: s.n.], 1987. P. 203–216.

DI PIETRO, Roberto; DURANTE, Antonio; MANCINI, Luigi; PATIL, Vishwas. Addressing the shortcomings of one-way chains. In: PROCEEDINGS of the 2006 ACM Symposium on Information, computer and communications security. [S.l.: s.n.], 2006. P. 289–296.

DUCAS, Léo. Accelerating Bliss: the geometry of ternary polynomials. **IACR Cryptol. ePrint Arch.**, v. 2014, p. 874, 2014.

DUCAS, Léo; KILTZ, Eike; LEPOINT, Tancrède; LYUBASHEVSKY, Vadim; SCHWABE, Peter; SEILER, Gregor; STEHLÉ, Damien. CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme. **IACR Transactions on Cryptographic Hardware and Embedded Systems**, v. 2018, n. 1, p. 238–268, Feb. 2018. DOI: 10.13154/tches.v2018.i1.238-268. Available from: https://tches.iacr.org/index.php/TCHES/article/view/839.

EATON, Edward. **Signature schemes in the quantum random-oracle model**. 2017. MA thesis – University of Waterloo.

EL BANSARKHANI, Rachid; BUCHMANN, Johannes. Improvement and Efficient Implementation of a Lattice-Based Signature Scheme. In: SELECTED Areas in Cryptography – SAC 2013, LNCS 8282. Berlin, Heidelberg: Springer, 2014. P. 48–67.

GARG, Sanjam; PANDEY, Omkant; SRINIVASAN, Akshayaram; ZHANDRY, Mark. **Breaking the Sub-Exponential Barrier in Obfustopia**. [S.l.: s.n.], 2016. Cryptology ePrint Archive, Report 2016/102. https://eprint.iacr.org/2016/102.

GENNARO, Rosario; HALEVI, Shai; RABIN, Tal. Secure hash-and-sign signatures without the random oracle. In: SPRINGER. INTERNATIONAL Conference on the Theory and Applications of Cryptographic Techniques. [S.l.: s.n.], 1999. P. 123–139.

GENTRY, Craig; PEIKERT, Chris; VAIKUNTANATHAN. Trapdoors for hard lattices and new cryptographic constructions. In: PROCEEDINGS of the fortieth annual ACM symposium on Theory of computing. [S.l.: s.n.], 2008. P. 197–206.

GOLDWASSER, Shafi; MICALI, Silvio; RIVEST, Ronald L. A digital signature scheme secure against adaptive chosen-message attacks. **SIAM Journal on computing**, SIAM, v. 17, n. 2, p. 281–308, 1988.

GORBUNOV, Sergey; VAIKUNTANATHAN, Vinod; WICHS, Daniel. Leveled fully homomorphic signatures from standard lattices. In: PROCEEDINGS of the forty-seventh annual symposium on Theory of computing. [S.l.: s.n.], 2015. P. 469–477.

IMPAGLIAZZO, Russel. A personal view of average-case complexity. In: IEEE. PROCEEDINGS of Structure inb Complexity Theory. Tenth Annual IEEE Conference. [S.l.: s.n.], 1995. P. 134–147.

KRAWCZYK, Hugo; RABIN, Tal. **Chameleon Hashing and Signatures**. [S.l.: s.n.], 1998. Cryptology ePrint Archive, Report 1998/010. `https://eprint.iacr.org/1998/010`.

LAMPORT, Leslie. Password authentication with insecure communication. **Communications of the ACM**, ACM New York, NY, USA, v. 24, n. 11, p. 770–772, 1981.

LANGLOIS, Adeline; STEHLÉ, Damien. Worst-case to average-case reductions for module lattices. **Designs, Codes and Cryptography**, Springer, v. 75, n. 3, p. 565–599, 2015.

LEGROW, Jason. **Post-quantum security of authenticated key establishment protocols**. 2016. MA thesis – University of Waterloo.

LI, BaoHong; LIU, YanZhi; YANG, Sai. Lattice-Based Universal Designated Verifier Signatures. In: IEEE. 2018 IEEE 15th Internationalç Conference on e-Business Engineering (ICEBE). [S.l.: s.n.], 2018. P. 329–334.

LU, Xingye; AU, Man Ho; ZHANG, Zhenfei. Raptor: A Practical Lattice-Based (Linkable) Ring Signature. In: DENG, Robert H.; GAUTHIER-UMAÑA, Valérie; OCHOA, Martín; YUNG, Moti (Eds.). **Applied Cryptography and Network Security**. Cham: Springer International Publishing, 2019b. P. 110–130.

LU, Xingye; AU, Man Ho; ZHANG, Zhenfei. Raptor: a practical lattice-based (linkable) ring signature. In: SPRINGER. INTERNATIONAL Conference on Applied Cryptography and Network Security. [S.l.: s.n.], 2019c. P. 110–130.

LYUBASHEVSKY, Vadim; NEVEN, Gregory. One-shot verifiable encryption from lattices. In: SPRINGER. ANNUAL International Conference on the Theory and Applications of Cryptographic Techniques. [S.l.: s.n.], 2017. P. 293–323.

MICCIANCIO, Daniele; PEIKERT, Chris. Trapdoors for lattices: Simpler, tighter, faster, smaller. In: SPRINGER. ANNUAL International Conference on the Theory and Applications od Cryptographic Techniques. [S.l.: s.n.], 2012. P. 700–718.

MOHASSEL, Payman. One-time signatures and chameleon hash functions. In: SPRINGER. INTERNATIONAL Workshop on Selected Areas in Cryptography. [S.l.: s.n.], 2010. P. 302–319.

PEIKERT, Chris. A decade of lattice cryptography. **Foundations and Trends® in Theoretical Computer Science**, Now Publishers Inc. Hanover, MA, USA, v. 10, n. 4, p. 283–424, 2016.

PEIKERT, Chris. Limits on the hardness of lattice problems in l-p norms. **Computational Complexity**, Springer, v. 17, n. 2, p. 300–351, 2008.

PLAMONDON, R.; SRIHARI, S.N. Online and off-line handwriting recognition: a comprehensive survey. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 22, n. 1, p. 63–84, 2000. DOI: 10.1109/34.824821.

PORNIN, Thomas. New Efficient, Constant-Time Implementations of Falcon. **IACR Cryptol. ePrint Arch.**, v. 2019, p. 893, 2019.

RIVEST, Ronald L; SHAMIR, Adi; TAUMAN, Yael. How to leak a secret. In: SPRINGER. INTERNATIONAL Conference on the Theory and Application of Cryptology and Information Security. [S.l.: s.n.], 2001. P. 512–565.

ROHLOFF, Kurt; COUSINS, Dave; POLYAKOV, Yuriy. **PALISADE Lattice Cryptography Library (release 1.9.2)**. [S.l.: s.n.], Apr. 2020. https://palisade-crypto.org/.

RÜCKERT, Markus. Adaptively secure identity-based identification from lattices without random oracles. In: SPRINGER. INTERNATIONAL Conference on Security and Cryptography for Networks. [S.l.: s.n.], 2010a. P. 345–362.

RÜCKERT, Markus; SCHNEIDER, Michael. Estimating the Security of Lattice-based Cryptosystems. **IACR Cryptol. ePrint Arch.**, Citeseer, v. 2010, p. 137, 2010.

SCHMIDT-SAMOA, Katja; TAKAGI, Tsuyoshi. Paillier's cryptosystem modulo p^2q and its applications to trapdoor commitment schemes. In: SPRINGER. INTERNATIONAL Conference on Cryptology in Malaysia. [S.l.: s.n.], 2005. P. 296–313.

SHAMIR, adi; TAUMAN, Yael. Improved online/offline signature schemes. In: SPRINGER. ANNUAL International Cryptology Conference. [S.l.: s.n.], 2001. P. 355–367.

SHOR, Peter W. Algorithms for quantum computation: discrete logarithms and factoring. In: IEEE. PROCEEDINGS 35th annual symposium on foundations of computer science. [S.l.: s.n.], 1994. P. 124–134.

SHOUP, Victor. Lower bounds for discrete logarithms and related problems. In: SPRINGER. INTERNATIONAL Conference on the Theory and Applications of Cryptographic Techniques. [S.l.: s.n.], 1997. P. 256–266.

STRONGSWAN. **Bimodal Lattice Signature Scheme (BLISS)**. [S.l.: s.n.], Feb. 2020. https://wiki.strongswan.org/projects/strongswan/wiki/BLISS. Accessed on 29/07/2020.

WANG, Xueli; CHEN, Yu; MA, Xuecheng. Adding linkability to ring signatures with one-time signatures. In: SPRINGER. INTERNATIONAL Conference on Information Security. [S.l.: s.n.], 2019. P. 445–464.

WIENER, Michael J. Bounds on Birthday Attack Times. **IACR Cryptology ePrint Archive**, Citeseer, v. 2005, p. 318, 2005.

XIE, Dong; PENG, Haipeng; LI, Lixiang; YANG, Yixian. Homomorphic signatures from chameleon hash function. **Informational Technology and Control**, v. 46, n. 2, p. 274–286, 2017.

ZHANDRY, Mark. **Cryptography in the Age of Quantum Computers**. 2015. PhD thesis – Stanford University.

ZHANG, Fangguo; SAVAFI-NAIN, Reihaneh; SUSILO, Willy. ID-Based Chameleon Hashes from Bilinear Pairings. **IACR Cryptol. ePrint Arch.**, Citeseer, v. 2003, p. 208, 2003.

ZHANG, Rui. Tweaking TBE/IBE to PKE transforms with chameleon hash functions. In: SPRINGER. INTERNATIONAL Conference on Applied Cryptography and Network Security. [S.l.: s.n.], 2007. P. 323–339.

ZIMMERMAN, Paul. **Factorization of RSA-250**. [S.l.: s.n.], 2020. Available from: `https://listserv.nodak.edu/cgi-bin/wa.exe?A2=NMBRTHRY;dc42ccd1.2002`.

Appendix

APPENDIX A – GENERAL FORKING LEMMA

To prove Theorem 26, we referred to the use of a theorem known as the general forking lemma, present in (BELLARE; NEVEN, Greagory, 2006). Here we show the full proof for this theorem as presented in the aforementioned work.

Recall that in Theorem 26 we had an adversary that would produce a list of q queries, gets a random and uniform response for each one, and in the end it would produce an output in which a choice for some specific response was present. We needed to know in that theorem what would happen if we run the adversary twice, using in both times the same source of randomness, repeating the same inputs to get the same results until the moment that the adversary send the same query whose response was chosen in the first execution. After this point we would replace the response for completely new random and uniform values. We needed to know the probability that at the same time this adversary suceeds, it chooses both times the same query in the final output and we send different responses in the chosen query in both executions.

A difficulty in this scenario is that these events are not necessarily independent. To compute these probabilities we should first abstract a little more this scenario. We can ignore without losing generality repeated queries that do not reveal new information to the adversary and as all the query responses are chosen uniformly at random, we also can ignore the queries, representing only their responses.

We can model the adversary $\mathcal{A}_2$ from Theorem 26 as an algorithm AdvA that takes as input some value pk (in our case, the public key) and a tuple with q elements $(h_1, \ldots, h_q)$ from some set H (representing the random responses for the queries) and outputs some value y (in the case of adversary $\mathcal{A}_2$ this value would be a forgery) and an integer $i \in [0, q]$. If $i > 0$, then this represents the scenario that $\mathcal{A}_2$ succeed at creating a forgery and did so using the response number i. If $i = 0$, this represents the scenario where the adversary failed, and there is no meaning in expecting a choice for some query.

The algorithm AdvA can make its choices using any arbitrary criteria, like adversary $\mathcal{A}_2$.

Using this algorithm AdvA to model adversary $\mathcal{A}_2$ from that security proof, we can create an algorithm that model adversary $\mathcal{B}_2$ as below:

- AdvB(pk):
  1. $(h_1, \ldots, h_q) \xleftarrow{\$} H^q$
  2. $(y, i) \xleftarrow{\$}$ AdvA$(pk, (h_1, \ldots, h_q))$
  3. **if** $i = 0$:
  4. **return** `fail`
  5. $(h'_i, \ldots, h'_q) \xleftarrow{\$} H^{q-i-1}$

6. $(y', i') \xleftarrow{\$} \text{ADVA}(pk, (h_1, \ldots, h_{i-1}, h'_i, \ldots, h'_q))$
7. **if** $i = i'$ **and** $h_i \neq h'_i$:
8. **return** `success`
9. **else**:
10. **return** `fail`

It runs algorithm ADVA twice, we assume that it does so feeding the algorithm with the same source of randomness. After the first execution, ADVA choose some input i, assuming that it suceeds. In the second execution, all the inputs h_j with $j < i$ are the same, but the other inputs are chosen uniformly at random. The algorithm returns `success` only if ADVB succeed both times, return the same choice for the input and the chosen inputs are different in both executions. This models the scenario in which adversary $\mathcal{B}_2$ succeeds.

Using the above algorithms, we can present the general forking lemma.

Theorem 30 ***(General Forking Lemma).*** *Let Ev_A be the event in which algorithm* ADVA *output $i > 0$ and let Ev_B the event in which algorithm* ADVB *outputs* `success`*. We have that:*

$$Pr[Ev_B] \geq Pr[Ev_A]\left(\frac{Pr[Ev_A]}{q} - \frac{1}{|H|}\right)$$

Proof: We have that:

$$\begin{aligned} Pr[Ev_B] = & \quad Pr[i = i' \wedge i > 0 \wedge h_i \neq h'_i] \\ \geq & \quad Pr[i = i' \wedge i > 0] - Pr[i > 0 \wedge h_i = h'_i] \\ = & \quad Pr[i = i' \wedge i > 0] - \frac{Pr[i>0]}{|H|} \\ = & \quad Pr[i = i' \wedge i > 0] - \frac{Pr[Ev_A]}{|H|} \end{aligned}$$

Now to prove that $Pr[i = i' \wedge i > 0] \geq \frac{Pr[Ev_A]^2}{q}$:

$$\begin{aligned} Pr[i = i' \wedge i > 0] = & \quad \sum_{j=1}^{q} Pr[i = j \wedge i' = j] \\ = & \quad \sum_{j=1}^{q} Pr[i = j] \cdot Pr[i' = j | i = j] \end{aligned}$$

Let for all $j \in [1, q]$, the following function represent the probability that for fixed constant inputs $(h_1, \ldots, h_{j-1})$ and for random coins ρ chosen from some set R, the probability that algorithm ADVA return j as first element in its output.

$$x_j(\rho, h_1, \ldots, h_{j-1}) = Pr[i = j | (h_j, \ldots, k_q) \xleftarrow{\$} H^{q-j-1}; (i, y) \xleftarrow{\$} \text{ADVA}(pk, (h_1, \ldots, h_q); \rho)]$$

Now we can rewrite the previous equation as:

$$Pr[i = i' \wedge i > 0] = \sum_{j=1}^{q} \sum_{\rho, h_1, \ldots, h_{j-1}} x_j(\rho, h_1, \ldots, h_{j-1}) \cdot X_i(\rho, h_1, \ldots, h_{j-1}) \cdot \frac{1}{|R||H|^{j-1}}$$

Let $E[x]$ be the expected value for some variable: the average of all possible values for x weighted by the probability of each value. This means that by definition we can express the inner sum of the above equation as $E[X_j^2]$ where x_j represents all possible outputs for the function $x_j()$:

$$Pr[i = i' \wedge i > 0] = \sum_{j=1}^{q} E[x_j^2]$$

Now we can use the fact that for all random variable x, we have that $E[x^2] \geq E[x]^2$. We present the proof for this statement directly after this proof. Now we have that:

$$Pr[i = i' \wedge i > 0] \geq \sum_{j=1}^{q} E[x_j]^2$$

For any real number x and integer $j \geq 1$, we have that $\sum_{j=1}^{q} x^2 = \frac{1}{q}(\sum_{j=1}^{q} x)^2$. We also show the proof for this statement after this proof. Using this information, we can write:

$$Pr[i = i' \wedge i > 0] \geq \frac{1}{q} \left(\sum_{j=1}^{q} E[x_j] \right)^2$$

For all $j \in [1, q]$, the value $E[x_j]$ is the probability that ADVA choose the specific j-th input value in its output. This means that $\sum_{j=1}^{q} E[x_j] = Pr[Ev_A]$. This means that:

$$Pr[i = i' \wedge i > 0] \geq \frac{(Pr[Ev_A])^2}{q}$$

Combining our results, we have proven the general forking lemma:

$$Pr[Ev_B] \geq \frac{(Pr[Ev_A])^2}{q} - \frac{Pr[Ev_A]}{|H|} = Pr[Ev_A] \left(\frac{Pr[Ev_A]}{q} - \frac{1}{|H|} \right)$$

■

In Theorem 26, we had $|H| = 2^n$ and we used q_{O2} to model the number of queries. We also have that $Pr[Ev_A] = DCMAadv^{RO}[\mathcal{A}_2, DSIG]$ and $Pr[Ev_B]$ is the probability that

adversary $\mathcal{B}_2$ succeeds at finding a forgery at SIG_{GPV} or a collision in the chameleon hash. This justifies the lower bound for $CMAadv^{RO}[\mathcal{B}_2, SIG_{GPV}]$ presented there.

Next we present the proofs of two assumptions made in the above proof for the general forking lemma:

Theorem 31

$$E[x^2] \geq E[x]^2$$

Proof: Let $y = E[x]$. As $(x-y)^2$ is non-negative:

$$\begin{aligned} 0 \leq \quad & E[(x-y)^2] = E[x^{-}2xy + y^2] = E[x^2] - 2yE[x] + y^2 \\ = \quad & E[x^2] - 2y^2 + y^2 = E[x^2] - y^2 = E[x^2] - E[x]^2 \end{aligned}$$

If $E[x^2] - E[x]^2 \geq 0$, then $E[x^2] \geq E[x]^2$. ■

Theorem 32 *Let $x_1, \ldots, x_q \in \mathbb{R}$ and $q \geq 1$. We have:*

$$\sum_{j=1}^{q} x_j^2 = \frac{1}{q}(\sum_{j=1}^{q} x_j)^2$$

Proof: We can treat x as a random variable that takes values from $\{x_1, \ldots, x_q\}$, each one with probability $1/q$. This means that:

$$E[x^2] = \frac{1}{q}\sum_{j=1}^{q} x_j^2$$

And:

$$E[x]^2 = \frac{1}{q^2}\left(\sum_{j=1}^{q} x_j\right)^2$$

By the previous theorem:

$$\frac{1}{q}\sum_{j=1}^{q} x_j^2 \geq \frac{1}{q^2}\left(\sum_{j=1}^{q} x_j\right)^2$$

Dividing by $1/q$ we get the desired result:

$$\sum_{j=1}^{q} x_j^2 \geq \frac{1}{q}\left(\sum_{j=1}^{q} x_j\right)^2$$

■

APPENDIX B – CHAMELEON HASH WITH STRONGER COLLISION RESISTANCE IN THE GENERIC GROUP MODEL

As seen in this dissertation, in some applications it is difficult to use chameleon hashes because while they are collision resistant primitives and they have a trapdoor to find collisions, not necessarily this means that we can publicize the computed collisions. The Attack Game 14 that defines the security of chameleon hashes assumes that the attacker does not have access to sample collisions.

To solve this problem, it is necessary an alternative and stronger definition of collision resistance. An attacker could obtain some sample collisions from an attacker, but should not be able to find new collisions after this.

Building a chameleon hash secure using this stronger definition of collision resistance is very challenging. Several previous works propose to generalize the concept of chameleon hash functions to meke easier to satisfy this requisite. In (ATENIESE; MEDEIROS, 2004) a chameleon hash construction was proposed to be used in such contexts. However, its security was proven in a non-standard model known as the generic group model.

This appendix presents the concept of stronger collision resistance for chameleon hash functions, describes the generic group model and presents the chameleon hash described in (ATENIESE; MEDEIROS, 2004). That chameleon hash also could be used to build preimage signatures, but it is not described in the main body of the dissertation because it is not secure against attackers with quantum algorithms. Solving the discrete logarithm problem, one could easily find collisions in this chameleon hash. Therefore, with a quantum computer with sufficient power, Shor's algorithm could be used to find collisions ((SHOR, 1994)).

B.1 STRONG COLLISION RESISTANCE

A stronger collision-resistance definition should ensure that even if an adversary is able to query sample collisions (and in the case of preimage chameleon hash functions, it should be able to query sample preimages), it should not be able to produce new collisions with values different than the ones revealed by the queries. We model this with the following attack game:

Attack Game 19 ***(Strong Collision Resistance).***

For a given preimage chameleon hash scheme CH defined over (M, R, D)*, and an adversary* $\mathcal{A}$*, consider a challenger and an adversary initialized with the security parameter* λ*.*

*The challenger runs the key generation algorithm CH.*KEYGEN(*lambda*) *to get* (ek, tk) *and sends ek to the adversary.*

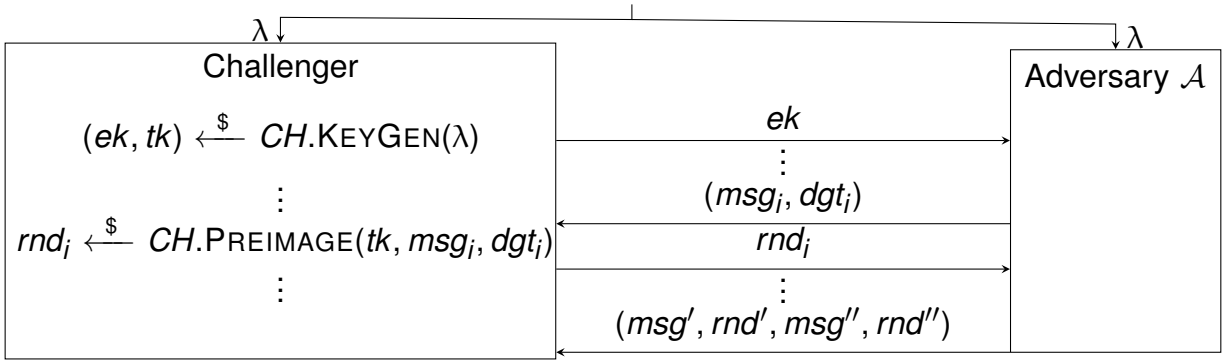

Figure 29 – Attack Game: Strong Collision Resistance

The adversary can perform a polynomial number of q_P preimage queries sending to the challenger pairs (msg_i, dgt_i) with $msg_i \in M$ and $dgt_i \in D$ and $i \in [1, q_S]$. For each of them, the challenger runs $rnd_i \xleftarrow{\$} CH.\text{PREIMAGE}(tk, msg_i, dgt_i)$ and sends rnd_i for the adversary.

In the end, the adversary outputs $(msg', rnd', msg'', rnd'')$ and we say that the adversary wins the game if $CH.\text{HASH}(ek, msg', rnd') = CH.\text{HASH}(ek, msg'', rnd'')$ and if for $i \in [1, q_S]$, we have that $(msg', rnd') \neq (msg_i, rnd_i)$ and $(msg'', rnd'') \neq (msg_i, rnd_i)$.

We denote by $SCRadv[\mathcal{A}, CH]$ the probability of a given adversary $\mathcal{A}$ win this attack game for the preimage chameleon hash CH.

B.2 GENERIC GROUP MODEL

This model was introduced at (SHOUP, 1997), initially as a way to study the limitations and lower bounds ofor generic algorithms: algorithms that solves mathematical problems without exploring knowledge present in the encoding of the elements.

In this model we take in consideration only generic attacks, attacks composed only by generic algorithms. This is usually used to create security proofs in cryptography when we use the discrete logarithm assumption.

The model assumes that an adversary can produce and find new values in a group performing a polynomially bounded number of multiplications. However, cannot extract any information about how each element from the group is encoded. We assume that all elements in our group are encoded randomly. Given two values, the only information possible to be extracted is if both elements are equal or not, as each element has a distinct encoding. Multiplications between known elements are made with the help of an oracle.

We will use the symbol $\sigma(x)$ to denote the encoding of an element x from the multiplicative group G. Furthermore, we denote $Enc[G]$ the set of all possible encodings of the group G. We will assume that encoding is a function that maps each element from G to a distinct number between 0 and $q-1$ where q is the order of the group. We could also assume that we can encode the elements to a larger set of numbers

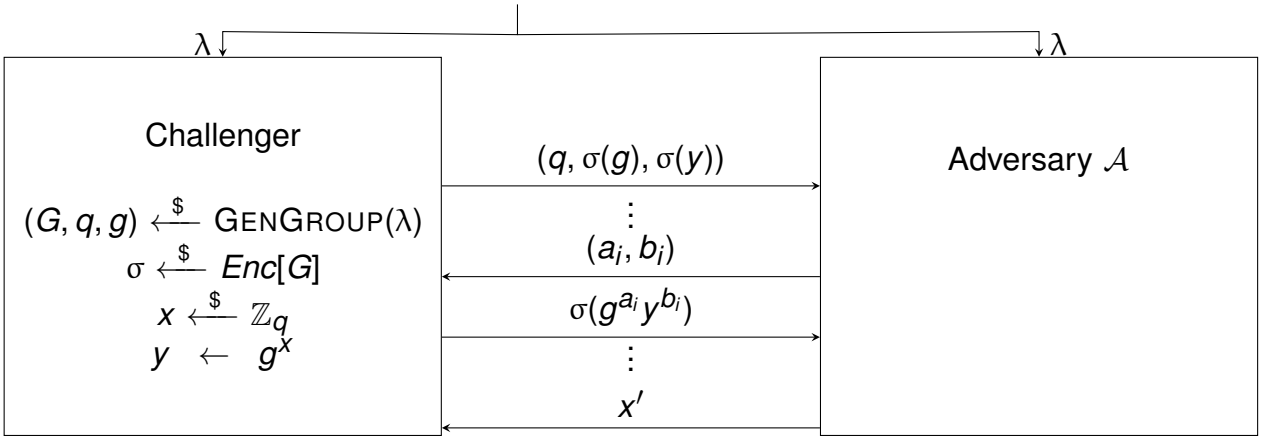

Figure 30 – Attack Game: Discrete Logarithm in Generic Group Model

or bitstrings, but we keep the encoding simple to simplify the security proofs and keep them more didactic.

Like with the random oracle model, we can take any attack game previously seen and update it to the generic group model. The difference is that the challenger generates a tuple (G, q, g) using algorithm GENGROUP described in the discrete logarithm assumption, sample a random and uniform encoding σ from $Enc[G]$ and each time it needs to send an element $x \in G$ to the adversary, it sends instead $\sigma(x)$. We can also send the adversary the order q of the group, but no other information about it.

We let the adversary make a polynomially bounded number of q_G group queries. For each query, it identifies two previously revealed encoded elements $\sigma(x), \sigma(y)$ and two exponents $a, b \in \mathbb{Z}_q$. The challenger sends $\sigma(x^a y^b)$ as response. If we transform the discrete logarithm to the generic group model, the attack game becomes as in the Figure 30.

Notice that in Figure 30 the adversary gets the encoding of elements g and y. Therefore, all other elements that it can obtain can be expressed as powers $g^a y^b$. The adversary wins the attack game if $g^x = y$, which is equivalent to $\sigma(g^x) = \sigma(y)$.

When we talk about the probability of winning an attacking game in the generic group model, we suffix our description with ^{GG}. For example, the probability that an adversary $\mathcal{A}$ wins the discrete logarithm attack game in the generic group model for group G is given by $DLadv^{GG}[\mathcal{A}, G]$.

We present in the next subsection an example of proof using the generic group model that also was presented in the paper that introduced the generic group model (SHOUP, 1997).

B.2.1 Proving the Discrete Logarithm Assumption in the Generic Group Model

Theorem 33 *In the generic group model, for all adversaries $\mathcal{A}$, the probability of solving the discrete logarithm problem in a group G with prime order q, denoted by*

$DLadv^{GG}[\mathcal{A}, G]$, is always negligible.

Proof: Proof in the generic group model usually are not security reduction proofs. We can prove exact and unconditional negligible upper bounds for the probabilities.

Notice that the real challenger is not efficient, as it needs to keep and apply a random and uniform encoding, which has a exponential space complexity for the same reasons as a random oracle, is not space efficient. Then, to prove that the discrete logarithm is a hard problem, we build a simulator that acts as a real challenger in this model, except with negligible probability. Then we prove that if our simulator succeeds at simulating a real challenger, no adversary can win the attack game interacting with the simulator, except with negligible probability.

Simulating a random encoding is not so different from simulating a random oracle. Our simulator can keep a dictionary with space to store up to $q_G + 3$ elements. Each element uses as key a pair of integers (a, b) modulo q that represent exponents and store the corresponding random encoding for $g^a y^b$.

In the first position, it generates and stores the key $(1, 0)$ and a random and uniform value that represents the encoding of $g^1 y^0 = g$ and in the second position we store the key $(0, 1)$ with the encoding randomly chosen that represents $g^0 y^1 = y$. These two are the encodings sent to the adversary during the initialization of the attack game. The next q_G elements will be generated for each group query sent by the adversary. Moreover, the last space in the dictionary is to generate a new encoding to check if the adversary output is correct and if the adversary wins the game. Like in the random oracle model, if an adversary query the same tuple (a, b) twice, we check that the query is repeated and send the same response again.

now we will change our simulator, making it different than a real challenger. Instead of choosing a random and uniform x at the beginning of the attack game and derive our encodings from real multiplications in the group, we will produce during the initialization a list with $q_G + 2$ distinct encodings and store them. For each new group query, we send as a response the next available encoding previously generated. After all the group queries, we choose a suitable value of x consistent with all previous responses.

This creates a limitation in our simulator. Interacting with a real challenger, an adversary $\mathcal{A}$ could find pairs $(a, b) \neq (c, d)$ such that $\sigma(g^a y^b) = \sigma(g^c y^d)$. However, in our simulator we are making all responses for group queries distinct if the pair of exponents are distinct. Therefore, there is a probability that our simulator fails at simulating a real challenger. It happens when the adversary find a collision in $f(a, b) = \sigma(g^a y^b)$, which happens if it finds values $a, b, c, d \in \mathbb{Z}_q$ such that $a + bx = c + dx$ for the random and uniform exponent x chosen by the challenger.

However, as we reveal to the adversary only random encodings, if the adversary tries to find such values of (a, b, d, c), the only information it learns with each query is

if it already found a collision or not. Not having any other information, the probability of finding a collision can be bounded by the same probability than a birthday attack, presented in Subsection 2.3.2. By the bounds found in that subsection, the probability of finding such collision is at most $\frac{p(p-1)}{4q}$ where p is the number of tried pairs, and q is the order of our group. In our attack game, we can produce and test at most $q_G + 2$ different pairs, including the initial pairs $(0, 1)$ and $(1, 0)$. Therefore, the upper bound for the probability of our simulator failing at simulating a real challenger is this birthday attack bound applied over $q_G + 3$ tries:

$$Pr[\text{Simulator fails}] \leq \frac{(q_G + 2)(q_G + 1)}{q}$$

Here, our group has a prime order. If not, then our group could contain subgroups and depending on the capacity of our adversary factor the order q, it could produce exponents such that all responses for the chosen group queries are elements from the same subgroup, which would increase the probability of collision. If the order is prime, this is not possible.

Making all these changes, here is a summary of how our simulator works:

- **Initialization:** The simulator is initialized with the security parameter λ. It obtains a tuple (G, q, g) invoking GENGROUP(λ). Next, it creates a list with random $q_G + 2$ distinct elements from $\mathbb{Z}_q$ representing the encoding of all possible group elements that we will generate.

  After this, it needs to choose suitable encodings for g and y to send to the adversary, as it needs these inputs to try to solve the discrete logarithm problem. The first element in the list is associated with $g^1 y^0 = g$. The simulator stores in a dictionary the exponents $(1, 0)$ and the chosen encoding.

  The second element in the list of encodings is associated with the element $g^0 y^1 = y$. Because of this, the simulator stores in the dictionary the pair of exponents $(0, 1)$ that represents y and the corresponding encoding, which is the second element in the list.

  Finally, it interacts with the adversary $\mathcal{A}$, sending $(q, \mathit{list}[1], \mathit{list}[2])$ that represents the order of the group and encodings $\sigma(g)$ and $\sigma(y)$.

- **Group Query:** When adversary $\mathcal{A}$ sends a group query (a_i, b_i), the simulator checks if this tuple of exponents is in the dictionary. If so, it returns the stored encoding associated with it. If not, the simulator chooses from $\mathit{list}[i]$ the next encoding. It stores this value in the dictionary, associating with the exponents (a_i, b_i) and sends it as a response.

- **Finalization:** After all the queries, the simulator chooses the exponent x that is chosen at the beginning of the attack game in a real challenger. It chooses some

consistent x that would not create a collision given all the group queries previously done. For all different combinations of distinct queried pairs (a_i, b_i) and (a_j, b_j), we must have that $a_i + b_i x \neq a_j + b_j x$ in modulus q. After choosing the value of x uniformly at random between all such consistent candidates, the adversary gets from the adversary $\mathcal{A}$ a value x' and this adversary wins the game ix $x = x'$.

Now assuming that this simulator succeed at simulating a real challenger, we will find the upper bound for the probability any adversary $\mathcal{A}$ succeed.

If the adversary had not performed any query, then its probability of success would be given by $1/q$. As it performs queries, for each two pair of queries (a_i, b_i), (a_j, b_j), it can deduce that $a_i + b_i x \neq a_j + b_j x$ and this would inform that $(a_i - a_j)(b_j - b_i)^{-1} \mod q$ is not a correct value for x. For the best case for $\mathcal{A}$, each pair of group queries would disclose to $\mathcal{A}$ the information that a new element is not a correct guess for the value of x.

The number of distinct combinations of two queried pairs is given by $\frac{(q_G+2)(q_G+1)}{2}$. Therefore, the probability of any adversary succeeds guessing the correct value of x would be bounded by $\frac{1}{q-(q_G+2)(q_G+1)/2}$.

Consider the following events:

- We will denote by $Pr[NotSim]$ the probability that our simulator fails at simulating correctly a real challenger. We checked that this happens only when the adversary find two pairs of colliding exponents for $\sigma(g^a y^b)$. Therefore, $Pr[NotSim] \leq \frac{(q_G+2)(q_G+1)}{q}$.
- We will denote by $Pr[Sim]$ the probability that our simulator correctly simulates a real challenger. This is $(1 - Pr[NotSim])$.
- We denote by $Pr[AdSim]$ the probability that an adversary wins against a simulator. We checked that $Pr[AdSim] \leq \frac{1}{q-(q_G+2)(q_G+1)/2}$.
- We denote by $Pr[AdChal]$ the probability that an adversary wins against a real challenger.

Therefore, we can create an upper bound for $DLadv^{GG}[\mathcal{A}, G]$ as:

$$DLadv^{GG}[\mathcal{A}, G] \leq Pr[NotSim]Pr[AdChal] + Pr[Sim]Pr[AdSim]$$

As we are interested in a upper bound, we can assume that $Pr[AdCHal] = 1$ and round $Pr[Sim]$ to 1:

$$DLadv^{GG}[\mathcal{A}, G] \leq \frac{(q_G + 2)(q_G + 1)}{q} + \frac{1}{q - (q_G + 2)(q_G + 1)/2}$$

We have that q_G is a value limited polynomially and q, the degree of the group superpolynomial. Therefore, all the above fractions on the right side of the inequation

represent negligible values. This means that the discrete logarithm problem for groups with prime orders is a hard problem on average in the generic group model. ■

B.2.2 Generic Group in a Post-Quantum Model

As the discrete logarithm is not a valid assumption against quantum adversaries, proofs in the generic group model are not valid in post-quantum models. The generic model fails to realistically address some possibilities of quantum adversaries, like sending queries with a superpolynomial number of exponents in superposition, for example.

This makes the generic group a helpful model only when considering classical adversaries, making it a less exciting model when compared with the random oracle model.

B.3 PREIMAGE CHAMELEON HASH FROM DISCRETE LOGARITHMS IN GENERIC GROUPS

This construction was first presented at (ATENIESE; MEDEIROS, 2004), where the authors were trying to address the problem of having the security of chameleon hash functions compromised if collisions are leaked. The construction can be proven secure using the stronger version of collision-resistance defined in Subsection B.1. However, to prove this it is necessary to use the generic group model as heuristics.

The chameleon hash CH_{GG} is defined over the sets (M, R, D), with the help of a multiplicative group G with prime order q treated as a generic group and using a hash function $HASH : M \times \mathbb{Z}_q \to \mathbb{Z}_q$ where $R = \mathbb{Z}_q \times \mathbb{Z}_q$ and $D = \mathbb{Z}_q$. As part of the generic group model, we assume that we will use a group G of order q (where q has λ bits) where its elements are represented in a completely random encoding as described in Subsection B.2.

We describe the chameleon hash in the generic group model as:

- CH_{GG}.KEYGEN(λ) :
  1. $(G, q, g) \xleftarrow{\$}$ GENGROUP(λ)
  2. $\sigma \xleftarrow{\$} Enc[G]$
  3. $x \xleftarrow{\$} \mathbb{Z}_q$
  4. $y \leftarrow g^x$
  5. $ek \leftarrow (q, \sigma(g), \sigma(y))$
  6. $tk \leftarrow (q, \sigma, x)$
  7. **return** (ek, tk)
- CH_{GG}.HASH$(ek, msg, (r_1, r_2))$:

1. $(q, \sigma(g), \sigma(y)) \leftarrow ek$
2. **return** $r_1 - \sigma(g^{r_2} y^{HASH(msg, r_1)}) \mod q$

- CH.PREIMAGE(tk, msg, d) :

1. $(q, \sigma, x) \leftarrow tk$
2. $k \xleftarrow{\$} \mathbb{Z}_q$
3. $r_1 \leftarrow d + \sigma(g^k) \mod q$
4. $r_2 \leftarrow k - x(HASH(msg, r_1)) \mod q$
5. **return** (r_1, r_2)

Note that in the generic group model an user always can compute CH.HASH performing a single group query to obtain $\sigma(g^{r_2} y^{HASH(msg, r_1)})$ after computing $HASH(msg, r_1)$. The trapdoor tk is never sent to an adversary in our security model, therefore it does not need to have its values encoded in the random encoding.

This construction works because if (r_1, r_2) was returned by CH.HASH(ek, msg, d), then:

$$\begin{aligned} CH.\text{HASH}(ek, msg, (r_1, r_2)) &= r_1 - \sigma(g^{r_2} y^{HASH(msg, r_1)}) \mod q \\ &= d + \sigma(g^k) - \sigma(g^{r_2} y^{HASH(msg, r_1)}) \mod q \\ &= d + \sigma(g^k) - \sigma(g^{r_2 + xHASH(msg, r_1)}) \mod q \\ &= d + \sigma(g^k) - \sigma(g^{k - xHASH(msg, r_1) + xHASH(msg, r_1)}) \mod q \\ &= d + \sigma(g^k) - \sigma(g^k) \mod q \\ &= d \end{aligned}$$

Theorem 34 *The preimage chameleon hash CH_{GG} has the strong uniformity property.*

Proof: This is assured by r_1. If the pair (r_1, r_2) is chosen uniformly at random for a given ek and a pair of messages msg' and msg'', independent of the value of $-\sigma(g^{r_2} y^{HASH(msg, r_1)})$, we are adding to it the value r_1 chosen uniformly at random. We are performing a operation in an additive group modulo q and as long as one term of the sum is random and uniform, the result also will be uniform and random. We have exactly the same probability for all possible digests in this chameleon hash as required by the strong uniformity property. ■

Theorem 35 *The chameleon hash CH_{GG} have strong collision-resistance property in the generic group model if HASH is collision-resistant.*

To prove the full collision-resistance in the generic group model, we modify the Attack Game 19 allowing the adversary to make q_P preimage queries, as already described in that attack game, and q_G group queries, as required by the generic group model.

With the group queries the adversary can send a pair of exponents (a_i, b_i) and gets as response $\sigma(g^{a_i} y^{b_i})$. As the adversary gets in the key *ek* the values $\sigma(g)$ and $\sigma(y)$, this query represents all the values in the group G that the adversary can obtain using only generic operations.

Like in the proof from Subsection B.2.1, we will build an efficient simulator for the challenger in the Attack Game 19 in the generic group model. We will show how this simulator succeeds in acting exactly like a legitimate challenger, except with negligible probability. And assuming that the simulator correctly simulate a legitimate challenger, we show how any adversary can succeed in its attack game only with negligible probability.

Our simulator works in the following way:

- **Initialization:**
  1. $(G, q, g) \xleftarrow{\$} \mathcal{G}(n)$
  2. Initialize an array **s** with $q_g + q_p + 4$ random and uniform elements from $\mathcal{Z}_q$.
  3. Initialize an empty dictionary that will store pairs of exponents (a_i, b_i) with the corresponding value attributed to $\sigma(g^{a_i} y^{b_i})$.
  4. Initialize an empty list for tuples sent in preimage queries. We will store tuples in this list for each preimage query and will use this list to know if in the end the adversary $\mathcal{A}$ sent a proper collision that uses no preimage queries results.
  5. Store in the dictionary the pair $(1, 0)$ with the element **s**[1]. This will represent the encoding for $g^1 y^0 = g$.
  6. Store in the dictionary the pair $(0, 1)$ with the element **s**[2]. This will represent the encoding for $g^0 y^1 = y$.
  7. Send $ek = (q, \mathbf{s}[1], \mathbf{s}[2])$ to the adversary.
- **Group Query:** When adversary $\mathcal{A}$ send the query (a_i, b_i), we check in the dictionary if this pair of exponents were already queried. If so, we send the same response sent before. If not, we choose as response the value **s**$[i + 2]$. We store this value in the dictionary with the pair (a_i, b_i) and send it as response to the adversary.
- **Preimage Query:** In the line 4 of algorithm CH_{GG}.PREIMAGE, notice that the value r_2 is chosen from a distribution uniform and random, since the value k is

uniform and random. This means that we can correctly simulate that algorithm first choosing r_2 uniformly at random and then choosing an appropriate r_1 to make the pair (r_1, r_2) correct. As we control the random encoding simulation, we can compute these values in the reverse order:

1. Upon receiving the preimage query (msg_i, d_i), choose $r_2 \xleftarrow{\$} \mathbb{Z}_q$.
2. Set $r_1 \leftarrow d_i + \mathbf{s}[i + q_G + 2] \pmod{q}$.
3. Store the pair of exponents $(r_2, HASH(msg_i, r_1))$ in the dictionary with the encoding value $\mathbf{s}[i + q_G + 2]$. If that pair of exponents is already present in the dictionary, the simulator failed and halts.
4. Store the values (msg_i, r_1, r_2) in the list that stores tuples from preimage queries. These values cannot be used in the final output of a collision.
5. Send $rnd_i = (r_1, r_2)$ as response to the challenger.

- **Finalization:** Now the simulator got the output $(msg', (r'_1, r'_2), msg'', (r''_1, r''_2))$. To know if the adversary won, it needs to compute CH.HASH for these values:

  1. Check in the dictionary if we define an encoding for the exponents $(r'_2, HASH(msg', r'_1))$. If not, then we associate $\mathbf{s}[q_g + q_p + 3]$ as the encoding for $g^{r'_2} y^{HASH(msg', r'_1)}$. We will denote this encoding by e'.
  2. Check in the dictionary if we define an encoding for the exponents $(r''_2, HASH(msg'', r''_1))$. If not, then we associate $\mathbf{s}[q_g + q_p + 4]$ as the encoding for $g^{r''_2} y^{HASH(msg'', r''_1)}$. We will denote this encoding by e''.
  3. The adversary wins if $r'_1 - e' \equiv r''_1 - e'' \pmod{q}$ and if neither (msg', r'_1, r'_2) nor (msg'', r''_1, r''_2) are stored in the list of tuples sent in preimage queries.

This simulator can fail in two different scenarios. Like the simulator from Subsection B.2.1, it assumes that the adversary will not find colliding pairs $(a_i, b_i) \neq (a_j, b_j)$ such that $\sigma(g^{a_i} y^{b_i}) = \sigma(g^{a_j} y^{b_j})$. Using the bound found in that subsection, here the probability of finding such colliding exponents in a group or preimage query or during the verification step is given by:

$$Pr[\mathcal{A} \text{ finds colliding exponents}] \leq \frac{(q_G + q_P + 4)(q_G + q_P + 3)}{q}$$

This will be a negligible probability because in the numerator both q_G and q_P are polynomially bounded values and q in denominator is a value that grows exponentially with the security parameter λ.

The second event in which the simulator can fail is if during the preimage query, when performing step 3, the pair of exponents $(r_2, HASH(msg_i, r_1))$ is already stored in the dictionary because it was explicitly queried in a group query or because it was

randomly produced in a previous preimage query. As the value r_2 is chosen uniformly at random in the preimage query, we can use this to find an upper bound for the probability of this happening.

In the worst case for our simulator, we can assume that an adversary $\mathcal{A}$ already sent all q_G group queries, all of them with distinct values a_i in the pair (a_i, b_i) and already sent all preimages queries except the last one, where each of them choose a distinct value for r_2. This means that while producing the response for the last preimage query the simulator has a probability of $(q_g+q_P+1)/q$ of failing and this is the worst-case probability for a single preimage query. As we have q_P preimage queries, the upper bound for the probability of our simulator failing during one of these queries is given by:

$$Pr[\text{Simulator fails in preimage query}] \leq \frac{q_P(q_g + q_P + 1)}{q}$$

This probability is also negligible because q grows exponentially and bot q_P and q_s are polynomially bounded. Combining the probability that the adversary finds colliding pairs of exponents with the probability that our simulator fails during a preimage query, we have the upper bound for the probability that our simulator fails to act as a legitimate challenger:

$$Pr[\text{Simulator fails}] \leq \frac{(q_G + q_P + 4)(q_G + q_P + 3)}{q} + \frac{q_P(q_g + q_P + 1)}{q}$$

Now we will find the upper bound for the probability that any adversary $\mathcal{A}$ win the strong collision-resistance attack game when interacting with our simulator, assuming that it succeeds at simulation.

Any adversary $\mathcal{A}$ can output a final collision following one of there three mutually exclusive strategies, assuming for now that it does not find collisions in *HASH*. The three strategies are:

- The adversary produced a collision $(msg', (r'_1, r'_2))$ and $(msg'', (r''_1, r''_2))$ such that it never queried and do not know the value for $\sigma(g^{r'_2} y^{HASH(msg', r'_1)})$ and $\sigma(g^{r''_2} y^{HASH(msg'', r''_1)})$. We will denote by $\mathcal{A}_1$ any adversary that follows this strategy.

- The adversary produced a collision $(msg', (r'_1, r'_2))$ and $(msg'', (r''_1, r''_2))$, but it queried and knows only one of the two encodings associated with the collision. We will assume that i this case it knows only the encoding $\sigma(g^{r''_2} y^{HASH(msg'', r''_1)})$ and will denote by $\mathcal{A}_2$ any adversary that uses this strategy.

- Adversary produced a collision $(msg', (r'_1, r'_2))$ and $(msg'', (r''_1, r''_2))$ and it knows both encodings associated with these two inputs. We will denote by $\mathcal{A}_3$ any adversary that follows this strategy.

When adversary $\mathcal{A}_1$ send a forgery, he simulator checks in the finalization step if $r'_1 - \mathbf{s}[q_P + q_G + 3] \equiv r''_1 - \mathbf{s}[q_P + q_G + 4] \pmod{q}$. Each subtraction can have q

different outputs. In the best scenario for adversary $\mathcal{A}_1$, it could correctly guess that it is interacting with a simulator and that both $\mathbf{s}[q_P + q_G + 3]$ and $\mathbf{s}[q_P + q_G + 4]$ will be distinct and different than all previous encodings.

Using this strategy, adversary $\mathcal{A}_1$ could try to guess what are the two last values stored in $\mathbf{s}$ and would guess correctly with probability at most $1/(q-q_P-q_G-2)(q-q_P-q_G-3)$. After the guess, it chooses suitable values for r_1' and r_1'' to produce the collision. If it guess correctly, it wins the attack game. Even if it guess wrong, in the best case for it, there are q other q different possible values for $\mathbf{s}[q_P + q_G + 3]$ and $\mathbf{s}[q_P + q_G + 4]$ that also result in a collision for the chosen (r_1', r_2''). Therefore, the upper bound for the victory of $\mathcal{A}_1$ is given by:

$$Pr[\mathcal{A}_1 \text{ wins against simulator}] \leq \frac{q}{(q-q_P-q_G-2)(q-q_P-q_G-3)}$$

The above probability is negligible, as the numerator is q and the denominator can be written in the form $aq^2 + bq + c$ and q grows exponentially with the security parameter λ.

The upper bound for the probability of success for $\mathcal{A}_2$ can be found similarly. In this case, to verify if the adversary won the game, the simulator checks if $r_1' - e' \equiv r_1'' - \mathbf{s}[q_P + q_G + 4] \pmod{q}$ where e' is a value known by the adversary. In this case, the adversary can try to guess what is the value stored in $\mathbf{s}[q_P + q_G + 4]$ and would guess correctly at most with probability $1/(q - q_P - q_G - 2)$. It would then choose r_1' and r_1'' accordingly to produce a collision. However, if it guesses wrong, it would fail at producing a collision. Therefore, the probability of any adversary $\mathcal{A}_2$ winning the attack game against the simulator is given by:

$$Pr[\mathcal{A}_2 \text{ wins against simulator}] \leq \frac{1}{(q-q_P-q_G-2)}$$

The above probability in all scenarios is a greater value than the upper bound for $\mathcal{A}_1$, but still is a negligible probability.

For the adversary $\mathcal{A}_3$, after it produces the possible collision $(msg', (r_1', r_2'))$ and $(msg'', (r_1'', r_2''))$, the simulator test if it succeed checking if $r_1' - e' \equiv r_1'' - e'' \pmod{q}$ where $e' = \sigma(g^{r_2'} y^{HASH(msg', r_1')})$ and $e'' = \sigma(g^{r_2''} y^{HASH(msg', r_1'')})$. Moreover, we know that both e' and e'' are values previously returned by a group query. We know that they were not produced by a preimage query because this would mean that either the adversary is returning directly values returned by such queries in the collision, which is not allowed by the rules in this attack game, or the adversary found a collision in *HASH*, and for now we are assuming that our adversaries cannot find collisions.

As we are temporally assuming that adversaries cannot find collisions for *HASH*, this also means that for all q_G group queries in the form (a_i, b_i), the adversary knows at most a single pair (msg_i, r_{1i}) such that $HASH(msg_i, r_{1i}) = b_i$. And if such group query

is the one that produced encodings e' or e'' used in the final output, then necessarily the final output contains $(msg_i, (r_{1i}, b_i))$.

This means that we can find an upper bound for the probability of $\mathcal{A}_3$ succeeding at find a collision in CH_{GG} assuming that in the best case for $\mathcal{A}_3$, for all group queries (a_i, b_i) that return an encoding e_i, it knows a single preimage (msg_i, r_{1i}) such that $HASH(msg_i, r_{1i}) = b_i$ and checking the probability for having $r_{1i} - e_i \equiv r_{1j} - e_j \pmod{q}$ for any two group queries (a_i, b_i) and (a_j, b_j).

To find this probability, consider the scenario where $\mathcal{A}_3$ already made q_G–1 group queries, obtaining in all them different results for the subtraction of their corresponding r_{1i} and encoding e_i. When $\mathcal{A}_3$ is performing the last query, to succeed in the attack game, it needs to produce the last r_{1q_G} such that $r_{1q_G} - e_{q_G}$ is a result already obtained in a previous query. It can choose value r_{1q_G}, but will discover what is e_{q_G} only after it sends the query and receive a response.

If adversary $\mathcal{A}$ wants to manipulate the subtraction $r_{1q_G} - e_{q_G}$ to produce the same value than a single specific previous query, then this would be the same scenario found by adversary $\mathcal{A}_2$ when it trying to guess correct values for r'_1 when producing the last output, except that here adversary $\mathcal{A}_3$ performed one less query. In such case, the probability of guessing the correct r_{q_G} would be given by $1/(q - q_P - q_G - 1)$. However, in this case adversary $\mathcal{A}_3$ do not need to guess r_{q_G} such that the subtraction is equal to one specific previous value, it is enough if the subtraction is equal to any of the previous $(q_G - 1)$ values. In this case, the probability of finding a suitable collision using the last query is given by $(q_G - 1)/(q - q_P - q_G - 1)$.

That is only the probability for the last query. All the previous queries also have a smaller chance of producing a suitable collision. We can use the same bound for all q_G queries and get the upper bound for the probability of adversary $\mathcal{A}_3$ succeeds as the sum of the probabilities for each query:

$$Pr[\mathcal{A}_3 \text{ wins against simulator}] \leq \frac{q_G(q_G - 1)}{q - q_P - q_G - 1} \leq \frac{q_G^2}{q - q_P - q_G - 1}$$

For sufficient big values of q_G, this probability is greater than what was computed for $\mathcal{A}_1$ and $\mathcal{A}_2$, but still is a negligible probability. Therefore, we can use this as the upper bound for the probability of any adversary $\mathcal{A}$ succeed against the simulator. However, there is another possibility that we did not consider until now. We assumed that no adversary is able to find collisions in *HASH*. If one could find a single collision, it would be trivial to find a collision for the chameleon hash CH_{GG}. The probability of finding collisions in *HASH* can be represented by $CRadv[\mathcal{B}, HASH]$ for some adversary $\mathcal{B}$. Therefore, the probability for any adversary $\mathcal{A}$ winning the attack game when interacting with the simulator is given by:

$$Pr[\mathcal{A} \text{ wins against simulator}] \leq \frac{q_G^2}{q - q_P - q_G - 1} + CRadv[\mathcal{B}, HASH]$$

Finally, as we did in Subsection B.2.1, we can find an upper bound for any adversary $\mathcal{A}$ winning our attack game against a legitimate challenger as the probability of winning against the simulator plus the probability of the simulator failing to act as a real challenger. We have then that for all adversaries $\mathcal{A}$:

$$SCRadv^{GG}[\mathcal{A}, CH_{GG}] \leq \frac{(q_G + q_P + 4)(q_G + q_P + 3)}{q} + \frac{q_P(q_g + q_P + 1)}{q} + \frac{q_G^2}{q - q_P - q_G - 1} + CRadv[\mathcal{B}, HASH]$$

By our assumption, the function *HASH* is collision-resistant. Therefore, all values in the right side of the above inequation are negligible and any adversary $\mathcal{A}$ that tries to find a collision in CH_{GG} in the strong collision-resistance attack game has a negligible value as upper bound for its success.

■

As the above proof is in the generic group model, it gives no guarantees if we assume that the adversary can use non-generic algorithms. Quantum adversaries also can break the collision-resistance of this construction, as they can solve the discrete logarithm problem and so they can easily find the trapdoor key *tk* given an evaluation key *ek*.

The construction presented in (ATENIESE; MEDEIROS, 2004) suggest using as a generic group the subgroup of quadratic residues modulo $p = (2q+1)$ such that both q and $2q+1$ are primes. With such initialization, the keys are represented as $ek = (q, g, y)$ and $tk = (q, x)$, such that $g^x \mod p = y$. The order of the group is q. The algorithm CH_{GG}.HASH is instantiated as:

- CH_{GG}.HASH$(ek, msg, (r_1, r_2))$:
  1. $(q, g, y) \leftarrow ek$
  2. **return** $r_1 - (g^{r_2} y^{HASH(msg, r_1)} \mod p) \mod q$

  And the CH_{GG}.PREIMAGE algorithm is instantiated as:

- CH.PREIMAGE(tk, msg, d) :
  1. $(q, x) \leftarrow tk$
  2. $k \xleftarrow{\$} \mathbb{Z}_q$
  3. $r_1 \leftarrow d + (g^k \mod p) \mod q$
  4. $r_2 \leftarrow k - x(HASH(msg, r_1)) \mod q$
  5. **return** (r_1, r_2)

One detail of this instantiation is that the group has order q, a number with λ bits while elements of this group are encoded using $\lambda + 1$ bits, as they are a group of quadratic residues modulo $p = 2q + 1$. This means that it could be possible to find two different elements y_1 and y_2 in the group such that $y_1 \mod q = y_2 \mod q$. This possibility was not modeled by our security proof because we assumed that the encoding of our group has the same number of bits than its order. This doubles the upper bound for the probability of finding collisions, but other than that, the security proof remains valid and the upper bounds remains asymptotically the same.

Annex

ANNEX A – SYSTEMATIC REVIEW ABOUT PREIMAGE CHAMELEON HASHES

In this annex we describe the stages of our systematic review about the use of preimage chameleon hashes in the literature.

A.1 INTRODUCTION

Preimage chameleon hashes, or chameleon hashes with a more powerful trapdoor which allows for preimage computation, are not a new concept, but they are used very rarely in the literature. When we thought about the idea of using this kind of chameleon hash to build preimage signatures, we conducted a systematic review to investigate which authors already explored this concept and what use cases of this construction were already proposed.

A challenge present is that preimage cameleon hashes, while noticed by different researchers, had not a standard name. In some papers they simply define as “chameleon hashes” what here we call “preimge chameleon hashes”. In other papers, they call it “a slightly generalization”, without naming it. Finally, in (LU et al., 2019b), the name “Chameleon Hash Plus” was proposed, but very few authors adopted this name.

Despite this difficuly, one advantage is that chameleon hashes are a relatively new cryptographic primitive, proposed first in 1998. So, even if we search in the literature for keywords that in other areas could be too generic, the quantity of results usually is tractable. Despite the lack of a standard name, we searched for preimage chameleon hashes looking for articles about chameleon hashes and which explored preimage properties.

A.2 OBJECTIVE

Our objective initially was discover if our idea about a preimage chameleon hash scheme was a new propose and, if not, discover for what purposes this primitive was already being proposed. We was specially interested about possible uses in signature schemes to check if our idea about preimage signatures was not already introduced.

A.3 QUESTIONS

To fulfill the objective, we formulated two questions to answer with our review:

Q.1) The concept of a chameleon hash whose trapdoor allows for computing preimages instead of just second-preimages is already present in the literature?

Q.2) If so, for what kind of applications this primitive is being proposed?

These two questions are in fact interleaved. Finding the first article mentioning preimage chameleon hashes, the first question is answered positively and what this and the following articles present are answers for the second question. If no article

about preimage chameleon hashes were found, the second question would not need to be answered.

A.4 SELECTION OF FONTS

To select our fonts, we choosed the following keywords: “chameleon hash” and “preimage”. We choosed to look for these terms in all the content of each font, not only in the title or abstract.

We choosed as population published articles, thesis, dissertations and patents. We excluded books because needing to read and review entire books could be prohibitive.

To select our fonts, the first restriction is that the article should be available by a public link in the Internet or accessing the university network. We also limited our database to IEEEXplore, Google Scholar, ACM DL and Springer. We choosed only databases which allows for full-text search. We also restricted the language to english. And the search was limited to works published between 2010 and half of 2020.

This range of years was chosen because the first chameleon hash based on post-quantum assumptions was proposed on 2010 and the first papers that propose building signatures schemes with chameleon hashes were newer than this. Therefore, using the selected range of years we expected that we would not miss relevant articles to our research. The range already encompasses the majority of fonts found with our keyword. There are less than 30 fonts excluded by our search for being published before 2010.

A.4.1 Details About the Search

Library:	IEEEXPlore		
Date:	August 14,2020	**Period:**	2010–2020
URL:	https://ieeexplore.ieee.org/search/advanced		
Search String:	`(("Full Text & Metadata":"chameleon hash") AND "Full Text & Metadata":preimage)`		
Results:	5		

Library:	Google Scholar		
Date:	August 14,2020	**Period:**	2010–2020
URL:	https://scholar.google.com		
Search String:	`preimage AND "chameleon hash"`		
Results:	177		

Library:	ACM DL		
Date:	August 14,2020	**Period:**	2010–2019

URL:	https://dl.acm.org/search/advanced
Search String:	`[All: "chameleon hash"] AND [All: preimage] AND [Publication Date: (01/01/2010 TO 12/31/2019)]`
Results:	2

Library:	Springer		
Date:	August 14,2020	**Period:**	No option to restrict
URL:	https://link.springer.com/search		
Search String:	`"chameleon hash" AND preimage`		
Results:	55		

A.5 RESULTS

After joining all the results and excluding the duplicates and false positives, we got 149 initial results.

Next, we proceeded to keep only fonts which describe preimage chameleon hashes. For this, we read the abstract for all these results and also searched in the document for occurences of the terms “chameleon hash” and “preimage”. As expected given the amplitude of our search, the majority of the fonts were discarded in this second part. Only 21 papers and thesis were kept after this filter.

The next stage was read with more details all the content of the remaining fonts. After this third stage of more careful reading, only 12 works remained. All the discarded and accepted works are listed and classified below.

A.5.1 Discarded Results

1	GOLDSCHLAG, David M. et al. Temporarily hidden bit commitment and lottery applications. **International Journal of Information Security**, v. 9, n. 1, p. 33–50, Jan. 2010. ISSN 16155262. DOI: 10.1007/s10207-009-0094-1.
2	RÜCKERT, Markus. Strongly unforgeable signatures and hierarchical identity-based signatures from lattices without random oracles. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2010b. P. 182–200. DOI: 10.1007/978-3-642-12929-2_14.
3	LIU, Joseph K et al. Short generic transformation to strongly unforgeable signature in the standard model. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2010. P. 168–181. DOI: 10.1007/978-3-642-15497-3_11. Available from: https://link.springer.com/chapter/10.1007/978-3-642-15497-3%7B%5C_%7D11.
4	ABE, Masayuki et al. Efficient hybrid encryption from ID-based encryption. **Designs, Codes, and Cryptography**, v. 54, n. 3, p. 205–240, Mar. 2010. ISSN 09251022. DOI: 10.1007/s10623-009-9320-0.

5	HANAOKA, Goichiro et al. Sequential Bitwise Sanitizable Signature Schemes. **IEICE TRANS. FUNDAMENTALS**, n. 1, 2011. DOI: 10.1587/transfun.E94.A.392. Available from: https://search.ieice.org/bin/summary.php?id=e94-a%7B%5C_%7D1%7B%5C_%7D392.
6	LI, Nan. **Self-certified digital signatures**. [S.l.], 2011. Available from: http://ro.uow.edu.au/theses/3404.
7	MOHASSEL, Payman. One-time signatures and chameleon hash functions. In: SPRINGER. INTERNATIONAL Workshop on Selected Areas in Cryptography. [S.l.: s.n.], 2010. P. 302–319.
8	LIU, Shengli et al. General construction of chameleon all-but-one trapdoor functions. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2011. P. 257–265. DOI: 10.1007/978-3-642-24316-5_18.
9	BOYLE, Elette et al. Fully leakage-resilient signatures. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2011. P. 89–108. DOI: 10.1007/978-3-642-20465-4_7.
10	UKNOWLEDGE, Uknowledge; CHANDRASEKHAR, Santosh. **CONSTRUCTION OF EFFICIENT AUTHENTICATION SCHEMES CONSTRUCTION OF EFFICIENT AUTHENTICATION SCHEMES USING TRAPDOOR HASH FUNCTIONS USING TRAPDOOR HASH FUNCTIONS**. [S.l.], 2011. Available from: https://uknowledge.uky.edu/gradschool%7B%5C_%7Ddiss/162.
11	YANG, Bo et al. **An Efficient Identity-based Signature from Lattice in the Random Oracle Model**. v. 7. [S.l.], 2011. P. 3963–3971. Available from: http://www.jofcis.com1553-9105/.
12	HARALAMBIEV, Kristiyan. **Efficient cryptographic primitives for non-interactive zero-knowledge proofs and applications**. 2011. PhD thesis – Citeseer.
13	CHASE, Melissa; KOHLWEISS, Markulf. A Domain Transformation for Structure-Preserving Signatures on Group Elements. **IACR Cryptology ePrint Archive**, v. 2011, p. 342, 2011. Available from: https://pdfs.semanticscholar.org/4bcd/5b423c9cef7a21701d7f9d0e6d251c8b072e.pdf%20http://dblp.uni-trier.de/db/journals/iacr/iacr2011.html%7B%5C#%7DChaseK11a.
14	KRZYWIECKI, Lukasz et al. Stamp and extend - Instant but undeniable timestamping based on lazy trees. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2012. P. 5–24. DOI: 10.1007/978-3-642-35371-0_2.
15	FAUST, Sebastian et al. **No Title**. 7658 LNCS. [S.l.: s.n.], 2012. P. 98–115. ISBN 9783642349607. DOI: 10.1007/978-3-642-34961-4_8. Available from: http://eprint.iacr.org/2012/045.
16	LI, Jin et al. Generic security-amplifying methods of ordinary digital signatures. **Information Sciences**, v. 201, p. 128–139, 2012. ISSN 00200255. DOI: 10.1016/j.ins.2012.03.006. Available from: https://www.sciencedirect.com/science/article/pii/S0020025512002058.

17	TAN, Xiao; WONG, Duncan S. Generalized first pre-image tractable random oracle model and signature schemes. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2012. P. 247–260. DOI: 10.1007/978-3-642-31448-3_19. Available from: https://link.springer.com/chapter/10.1007/978-3-642-31448-3%7B%5C_%7D19.
18	NACCACHE, David; POINTCHEVAL, David. Autotomic signatures. **Lecture Notes in Computer Science**, 6805 LNCS, p. 143–155, 2012. ISSN 03029743. DOI: 10.1007/978-3-642-28368-0_12.
19	HOFHEINZ, Dennis. All-but-many lossy trapdoor functions. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2012. P. 209–227. DOI: 10.1007/978-3-642-29011-4_14.
20	YANG, Bo et al. A strong designated verifier signature scheme with secure disavowability. In: PROCEEDINGS of the 2012 4th International Conference on Intelligent Networking and Collaborative Systems, INCoS 2012. [S.l.: s.n.], 2012. P. 286–291. DOI: 10.1109/iNCoS.2012.24. Available from: https://ieeexplore.ieee.org/abstract/document/6337932/.
21	CHASE, Melissa; KOHLWEISS, Markulf. A new hash-and-sign approach and structure-preserving signatures from DLIN. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2012. P. 131–148. DOI: 10.1007/978-3-642-32928-9_8. Available from: https://link.springer.com/chapter/10.1007/978-3-642-32928-9%7B%5C_%7D8.
22	TA, Nguyen; KHOA, Toan. **ZERO-KNOWLEDGE PROOF SYSTEMS FOR LATTICE-BASED CRYPTOGRAPHY**. [S.l.], 2013. Available from: https://www.ntu.edu.sg/home/khoantt/pubs/Khoa-PhDthesis.pdf.
23	BRESSON, Emmanuel et al. Off-line/on-line signatures revisited: A general unifying paradigm, efficient threshold variants and experimental results. **International Journal of Information Security**, v. 12, n. 6, p. 439–465, Nov. 2013. ISSN 16155262. DOI: 10.1007/s10207-013-0200-2.
24	HOFHEINZ, Dennis. **No Title**. 7881 LNCS. [S.l.: s.n.], 2013. P. 520–536. ISBN 9783642383472. DOI: 10.1007/978-3-642-38348-9_31. Available from: https://link.springer.com/chapter/10.1007/978-3-642-38348-9%7B%5C_%7D31.
25	BONEH, Dan; ZHANDRY, Mark. **No Title**. 8043 LNCS. [S.l.: s.n.], 2013. P. 361–379. ISBN 9783642400834. DOI: 10.1007/978-3-642-40084-1_21. Available from: https://link.springer.com/chapter/10.1007/978-3-642-40084-1%7B%5C_%7D21.
26	YAN, Jianhua et al. Efficient lattice-based signcryption in standard model. **Mathematical Problems in Engineering**, v. 2013, 2013. ISSN 1024123X. DOI: 10.1155/2013/702539. Available from: https://www.hindawi.com/journals/mpe/2013/702539/abs/.

27	RASS, Stefan. Dynamic proofs of retrievability from chameleon-hashes. In: ICETE 2013 - 10th International Joint Conference on E-Business and Telecommunications; SECRYPT 2013 - 10th International Conference on Security and Cryptography, Proceedings. [S.l.: s.n.], 2013. P. 296–304. DOI: 10.5220/0004505102960304. Available from: https://ieeexplore.ieee.org/abstract/document/7223178/?casa%7B%5C_%7Dtoken=ut9a7lFVYCEAAAAA:g7aXFZ71kMzPTnilW0eEFEYu0jOgtTsOPy%7B%5C_%7DdHwTcC-kIrVi1DoPD8tAUCPKixKJslqJPdtgX.
28	VELEMA, Maria. Classical Encryption and Authentication under Quantum Attacks, July 2013. arXiv: 1307.3753. Available from: http://arxiv.org/abs/1307.3753.
29	YANG, Bo et al. A novel construction of SDVS with secure disavowability. **Springer**, v. 16, p. 807–815, 2013. DOI: 10.1007/s10586-013-0254-y. Available from: https://link.springer.com/content/pdf/10.1007/s10586-013-0254-y.pdf.
30	ARANHA, Diego F. et al. The realm of the pairings. In: LECTURE Notes in Computer Science. [S.l.]: Springer Verlag, 2014. P. 3–25. DOI: 10.1007/978-3-662-43414-7_1.
31	ZHU, Yan et al. Secure and efficient random functions with variable-length output. **Journal of Network and Computer Applications**, v. 45, p. 121–133, 2014. ISSN 10958592. DOI: 10.1016/j.jnca.2014.07.033. Available from: https://www.sciencedirect.com/science/article/pii/S1084804514001817.
32	LIU, Shengli et al. Public-key encryption scheme with selective opening chosen-ciphertext security based on the Decisional Diffie-Hellman assumption. **Concurrency Computation Practice and Experience**, v. 26, n. 8, p. 1506–1519, 2014. ISSN 15320634. DOI: 10.1002/cpe.3021. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.3021.
33	RAINER, Sebastian; SAARBRÜCKEN, Gerling. **Plugging in trust and privacy: three systems to improve widely used ecosystems**. [S.l.], 2014. Available from: https://core.ac.uk/download/pdf/77125693.pdf.
34	BONEH, Dan; CORRIGAN-GIBBS, Henry. **LNCS 8873 - Bivariate Polynomials Modulo Composites and Their Applications**. [S.l.], 2014. Available from: https://link.springer.com/content/pdf/10.1007/s11390-007-9015-9.pdf.
35	QIN, Baodong; LIU, Shengli. Leakage-flexible CCA-secure public-key encryption: Simple construction and free of pairing. In: LECTURE Notes in Computer Science. [S.l.]: Springer Verlag, 2014. P. 19–36. DOI: 10.1007/978-3-642-54631-0_2.
36	WANG, Yuyu; TANAKA, Keisuke. Generic transformation to strongly existentially unforgeable signature schemes with leakage resiliency. **Lecture Notes in Computer Science**, Springer Verlag, v. 8782, p. 117–129, 2014. ISSN 16113349. DOI: 10.1007/978-3-319-12475-9_9.
37	RAMCHEN, Kim; WATERS, Brent. Fully Secure and Fast Signing from Obfuscation. **dl.acm.org**, Association for Computing Machinery, n. 1, p. 659–673, Nov. 2014. DOI: 10.1145/2660267.2660306. Available from: http://dx.doi.org/10.1145/2660267.2660306.

38	BÖHL, Florian et al. Confined Guessing: New Signatures From Standard Assumptions. **Journal of Cryptology**, Springer New York LLC, v. 28, n. 1, p. 176–208, 2014. ISSN 14321378. DOI: 10.1007/s00145-014-9183-z.
39	THAKUR, Tejeshwari; SHARMA, Birendra Kumar. CERTIFICATELESS CHAMELEON HASH SCHEME BASED ON RSA ASSUMPTION. **Math. Lett**, p. 12, 2014. ISSN 2049-9337. Available from: http://scik.org.
40	GHASEMI, Mohsen. An algorithmic-type classification of tetravalent one-regular graphs using computer algebra tools. **Fundamenta Informaticae**, v. 135, n. 3, p. 211–228, 2014. ISSN 01692968. DOI: 10.3233/FI-2014-1119. Available from: https://content.iospress.com/articles/fundamenta-informaticae/fi135-3-01.
41	REN, Wei; LIU, Yuliang. A lightweight possession proof scheme for outsourced files in mobile cloud computing based on chameleon hash function. **International Journal of Computational Science and Engineering**, v. 9, n. 4, p. 339–346, 2014. ISSN 17427193. DOI: 10.1504/IJCSE.2014.060715. Available from: https://www.inderscienceonline.com/doi/abs/10.1504/IJCSE.2014.060715.
42	YANG, Chunli et al. A fuzzy identity-based signature scheme from lattices in the standard model. **Mathematical Problems in Engineering**, v. 2014, 2014. ISSN 15635147. DOI: 10.1155/2014/391276. Available from: https://www.hindawi.com/journals/mpe/2014/391276/abs/.
43	SCHÖDER, Dominique; SIMKIN, Mark. Veristream – A framework for verifiable data streaming. In: LECTURE Notes in Computer Science. [S.l.]: Springer Verlag, 2015. P. 548–566. DOI: 10.1007/978-3-662-47854-7_34.
44	FROMKNECHT, Conner. **One-Time, Zero-Sum Ring Signature**. [S.l.], 2015. Available from: https://bitcoinedge.org/ja/papers/demo.pdf.
45	DER NATURWISSENSCHAFTEN, Doktors. **On Cryptographic Building Blocks and Transformations zur Erlangung des akademischen Grades eines**. [S.l.], 2015. Available from: https://d-nb.info/1078957703/34.
46	HUANG, Zhengan et al. n-Evasive all-but-many lossy trapdoor function and its constructions. **Security and Communication Networks**, v. 8, n. 4, p. 550–564, 2015. ISSN 19390122. DOI: 10.1002/sec.1002. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1002.
47	YAN, Jianhua et al. Identity-based signcryption from lattices. **Security and Communication Networks**, John Wiley and Sons Inc., v. 8, n. 18, p. 3751–3770, Dec. 2015. ISSN 19390122. DOI: 10.1002/sec.1297.
48	BLAZY, Olivier; CHEVALIER, Céline. Generic construction of UC-secure oblivious transfer. In: LECTURE Notes in Computer Science. [S.l.]: Springer Verlag, 2015. P. 65–86. DOI: 10.1007/978-3-319-28166-7_4.

49	KIM, Kee Sung; JEONG, Ik Rae. Efficient verifiable data streaming. **Security and Communication Networks**, v. 8, n. 18, p. 4013–4018, 2015. ISSN 19390122. DOI: 10.1002/sec.1317. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1317.
50	HABER, Stuart et al. **Efficient Transparent Redactable Signatures with a Single Signature Invocation Efficient Transparent Redactable Signatures with a Single Signature Invocation 4**. [S.l.], 2015. Available from: https://www.labs.hpe.com/techreports/2014/HPL-2014-90R1.pdf.
51	ZHAO, Xiufeng et al. Cloud data integrity checking protocol from lattice. **International Journal of High Performance Computing and Networking**, v. 8, n. 2, p. 167–175, 2015. ISSN 17400570. DOI: 10.1504/IJHPCN.2015.070020. Available from: https://www.inderscienceonline.com/doi/abs/10.1504/IJHPCN.2015.070020.
52	WANG, Zhiwei; YIU, Siu Ming. CCA secure PKE with auxiliary input security and leakage resiliency. In: LECTURE Notes in Computer Science. [S.l.]: Springer Verlag, 2015. P. 319–335. DOI: 10.1007/978-3-319-23318-5_18.
53	BOYEN, Xavier; LI, Qinyi. Towards tightly secure lattice short signature and Id-based encryption. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2016. P. 404–434. DOI: 10.1007/978-3-662-53890-6_14. Available from: https://link.springer.com/chapter/10.1007/978-3-662-53890-6%7B%5C_%7D14.
54	ALKIM, Erdem et al. **TESLA: Tightly-Secure Efficient Signatures from Standard Lattices**. [S.l.], 2016. Available from: https://www.cryptosith.org/papers/tesla-20161005.pdf.
55	NOH, Geontae; JEONG, Ik Rae. Strong designated verifier signature scheme from lattices in the standard model. **Security and Communication Networks**, v. 9, n. 18, p. 6202–6214, 2016. ISSN 19390122. DOI: 10.1002/sec.1766. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1766.
56	SABIR KIRAZ, Mehmet; UZUNKOL, Osmanbey. **Still Wrong Use of Pairings in Cryptography**. [S.l.], 2016. arXiv: 1603.02826v3. Available from: https://arxiv.org/abs/1603.02826.
57	LIBERT, Benoît et al. Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2016. P. 373–403. DOI: 10.1007/978-3-662-53890-6_13. Available from: https://hal.inria.fr/hal-01267123.
58	XIE, Dong et al. Short lattice signatures with constant-size public keys. **Security and Communication Networks**, v. 9, n. 18, p. 5490–5501, 2016. ISSN 19390122. DOI: 10.1002/sec.1712. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1712.
59	FOUAD, Ahmed; IBRAHIM, Shedeed. NEW SECURE SOLUTIONS FOR PRIVACY AND ACCESS CONTROL IN HEALTH INFORMATION EXCHANGE, 2016. DOI: 10.13023/ETD.2016.307. Available from: http://dx.doi.org/10.13023/ETD.2016.307.

60	THAKUR, Tejeshwari; KUMAR SHARMA, Birendra. International Journal of Applied Mathematics ID-BASED CHAMELEON HASHING AND CHAMELEON SIGNATURE BASED ON GQ SCHEME. **diogenes.bg**, v. 29, n. 2, p. 227–242, 2016. ISSN 1314-8060. DOI: 10.12732/ijam.v29i2.7. Available from: http://diogenes.bg/ijam/contents/2016-29-2/7/.
61	WANG, Yuyu; TANAKA, Keisuke. Generic transformations for existentially unforgeable signature schemes in the bounded leakage model. In: 12. SECURITY and Communication Networks. [S.l.: s.n.], 2016. P. 1829–1842. DOI: 10.1002/sec.1436. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1436.
62	DERLER, David et al. Digital Signatures from Symmetric-Key Primitives. **IACR Cryptology ePrint Archive**, v. 2016, p. 1085, 2016. Available from: https://www.technologyreview.com/s/600715/nsa-says%20http://dblp.uni-trier.de/db/journals/iacr/iacr2016.html%7B%5C#%7DDerlerORRS16.
63	KOSHY, Binoj et al. Chameleon salting: The new concept of authentication management. In: SMART Innovation, Systems and Technologies. [S.l.]: Springer Science and Business Media Deutschland GmbH, 2016. P. 323–337. DOI: 10.1007/978-3-319-30927-9_32.
64	THANALAKSHMI, P.; ANITHA, R. A graph-based chameleon signature scheme. In: SMART Innovation, Systems and Technologies. [S.l.]: Springer Science and Business Media Deutschland GmbH, 2016. P. 327–335. DOI: 10.1007/978-81-322-2529-4_34.
65	HUANG, Jianye et al. A black-box construction of strongly unforgeable signature schemes in the bounded leakage model. In: LECTURE Notes in Computer Science. [S.l.]: Springer Verlag, 2016. P. 320–339. DOI: 10.1007/978-3-319-47422-9_19.
66	HU, Xiaoming et al. Strong designated verifier signature schemes with undeniable property and their applications. **Security and Communication Networks**, v. 2017, 2017. ISSN 19390122. DOI: 10.1155/2017/7921782. Available from: https://www.hindawi.com/journals/scn/2017/7921782/abs/.
67	ALKIM, Erdem et al. **Revisiting TESLA in the quantum random oracle model**. [S.l.], 2017. Available from: https://link.springer.com/chapter/10.1007/978-3-319-28166-7%7B%5C_%7D4.
68	PAWLEGA, Filip et al. Revisiting TESLA in the Quantum Random. **PQCrypt 2017**, LNCS 10346, p. 143–162, 2017. DOI: 10.1007/978-3-319-59879-6.
69	BASTARA, Rpiit. Password Security Scheme based on a Honeywords Generation through Hashing Key and Salting Process. **academia.edu**, v. 5, n. 10, p. 195–199, 2017. Available from: http://www.academia.edu/download/55288754/IJSRDV5I100139.pdf.
70	LI, Qinyi. **Lattice Public-Key Encryption: Richer, Tighter, Stronger**. [S.l.], 2017. Available from: https://eprints.qut.edu.au/107651.
71	DÖTTLING, Nico; GARG, Sanjam. Identity-based encryption from the diffie-hellman assumption. In: LECTURE Notes in Computer Science. [S.l.]: Springer Verlag, 2017b. P. 537–569. DOI: 10.1007/978-3-319-63688-7_18.

72	GAO, Wen et al. Identity-based blind signature from lattices. **Wuhan University Journal of Natural Sciences**, v. 22, n. 4, p. 355–360, 2017. ISSN 10071202. DOI: 10.1007/s11859-017-1258-x. Available from: https://link.springer.com/chapter/10.1007/978-3-319-54705-3%7B%5C_%7D13.
73	WANG, Yuyu; TANAKA, Keisuke. Generic transformation for signatures in the continual leakage model. **IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences**, E100A, n. 9, p. 1857–1869, 2017. ISSN 17451337. DOI: 10.1587/transfun.E100.A.1857. Available from: https://search.ieice.org/bin/summary.php?id=e100-a%7B%5C_%7D9%7B%5C_%7D1857.
74	DÖTTLING, Nico; GARG, Sanjam. From Selective IBE to Full IBE and Selective HIBE. In: LECTURE Notes in Computer Science. [S.l.]: Springer Verlag, 2017a. P. 372–408. DOI: 10.1007/978-3-319-70500-2_13. Available from: https://link.springer.com/chapter/10.1007/978-3-319-70500-2%7B%5C_%7D13.
75	POETTERING, Bertram; STEBILA, Douglas. Double-authentication-preventing signatures. **International Journal of Information Security**, Springer Verlag, v. 16, n. 1, Feb. 2017. ISSN 16155270. DOI: 10.1007/s10207-015-0307-8.
76	CAMENISCH, Jan et al. Chameleon-hashes with ephemeral trapdoors and applications to invisible sanitizable signatures. In: LECTURE Notes in Computer Science. [S.l.]: Springer Verlag, 2017. P. 152–182. DOI: 10.1007/978-3-662-54388-7_6.
77	MA, Jinhua et al. An efficient and secure design of redactable signature scheme with redaction condition control. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2017. P. 38–52. DOI: 10.1007/978-3-319-57186-7_4. Available from: https://link.springer.com/chapter/10.1007/978-3-319-57186-7%7B%5C_%7D4.
78	WANG, Zecheng et al. Adaptive-ID Secure Identity-Based Signature Scheme from Lattices in the Standard Model. **IEEE Access**, v. 5, p. 20791–20799, 2017. ISSN 21693536. DOI: 10.1109/ACCESS.2017.2757464. Available from: https://ieeexplore.ieee.org/abstract/document/8053461/.
79	LIBERT, Benoît et al. Adaptive oblivious transfer with access control from lattice assumptions. In: LECTURE Notes in Computer Science. [S.l.]: Springer Verlag, 2017. P. 533–563. DOI: 10.1007/978-3-319-70694-8_19.
80	BOYEN, Xavier; LI, Qinyi. All-but-many lossy trapdoor functions from lattices and applications. In: SPRINGER. ANNUAL International Cryptology Conference. [S.l.: s.n.], 2017. P. 298–331.
81	CHEVALIER, Céline et al. **UC-Secure Protocols using Smooth Projective Hash Functions**. [S.l.], 2017. Available from: http://www.di.ens.fr/%7B~%7Dccheval/HdR/manuscrit.pdf.
82	WANG, Yujue et al. Verifiably encrypted cascade-instantiable blank signatures to secure progressive decision management. **International Journal of Information Security**, Springer Verlag, v. 17, n. 3, p. 347–363, June 2018. ISSN 16155270. DOI: 10.1007/s10207-017-0372-2.

83	MARTÍNEZ, Ramiro et al. **Title: Fully post-quantum protocols for e-voting, coercion resistant cast as intended and mixing networks Master of Science in Advanced Mathematics and Mathematical Engineering**. [S.l.], 2018. Available from: https://upcommons.upc.edu/handle/2117/113449.
84	EMMANUEL, Naina et al. **Structures and data preserving homomorphic signatures**. v. 102. [S.l.: s.n.], 2018. P. 58–70. DOI: 10.1016/j.jnca.2017.11.005. Available from: https://www.sciencedirect.com/science/article/pii/S1084804517303739.
85	ISHIZAKA, Masahito; MATSUURA, Kanta. Strongly Unforgeable Signature Resilient toÂ Polynomially Hard-to-Invert Leakage Under Standard Assumptions. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2018. P. 422–441. DOI: 10.1007/978-3-319-99136-8_23. Available from: https://link.springer.com/chapter/10.1007/978-3-319-99136-8%7B%5C_%7D23.
86	UZUNKOL, Osmanbey; KIRAZ, Mehmet Sabır. Still wrong use of pairings in cryptography. **Applied Mathematics and Computation**, Elsevier Inc., v. 333, p. 467–479, Sept. 2018. ISSN 00963003. DOI: 10.1016/j.amc.2018.03.062. arXiv: 1603.02826.
87	KEVIN, Atighehchi; MORGAN, Barbier. Signature Renewal for Low Entropy Data. In: PROCEEDINGS - 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications and 12th IEEE International Conference on Big Data Science and Engineering, Trustcom/BigDataSE 2018. [S.l.: s.n.], 2018. P. 873–884. DOI: 10.1109/TrustCom/BigDataSE.2018.00126. Available from: https://ieeexplore.ieee.org/abstract/document/8455994/.
88	JAGER, Tibor; KUREK, Rafael. Short Digital Signatures and ID-KEMs via Truncation Collision Resistance. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2018. P. 221–250. DOI: 10.1007/978-3-030-03329-3_8. Available from: https://link.springer.com/chapter/10.1007/978-3-030-03329-3%7B%5C_%7D8.
89	CHEN, Yu et al. Regular lossy functions and their applications in leakage-resilient cryptography. **Theoretical Computer Science**, v. 739, p. 13–38, 2018. ISSN 03043975. DOI: 10.1016/j.tcs.2018.04.043. Available from: https://www.sciencedirect.com/science/article/pii/S0304397518302937.
90	FLIUNT, Artem et al. **Redactable Blockchain-or-Rewriting History in Bitcoin and Friends**. [S.l.], 2018. Available from: https://pdfs.semanticscholar.org/f01c/15b90a5c33719cb452437adc9a5c60a917bb.pdf.
91	JAGADEESAN, Meena et al. **Proofs of Sequential Work**. [S.l.], 2018. Available from: https://www.boazbarak.org/cs127/Projects/seq%7B%5C_%7Dwork.pdf.
92	ZIMA, M. **P2P Cryptocurrency Exchange and Blockchain Size Reduction**. [S.l.], 2018. Available from: https://is.muni.cz/th/q1in8/thesis.pdf.
93	D2. **of the Art on Privacy-Enhancing Cryptography for Ledgers**. [S.l.], 2018. Available from: https://media.voog.com/0000/0042/1115/files/D2.1-State-of-the-Art-on-Privacy-Enhancing-Cryptography-for-Ledgers.pdf.

94	BLAZY, Olivier; CHEVALIER, Céline. Non-Interactive Key Exchange from Identity-Based Encryption. **dl.acm.org**, Association for Computing Machinery, v. 10, Aug. 2018. DOI: 10.1145/3230833.3230864. Available from: https://dl.acm.org/doi/abs/10.1145/3230833.3230864.
95	LIU, Jianwei et al. Multi-authority Fast Data Cloud-Outsourcing for Mobile Devices. **Springer**, Springer Verlag, 11060 LNCS, p. 231–249, 2018. DOI: 10.1007/978-3-319-99136-8_13. Available from: https://doi.org/10.1007/978-3-319-99136-8%7B%5C_%7D13.
96	WANG, Yuyu et al. Memory lower bounds of reductions revisited. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2018. P. 61–90. DOI: 10.1007/978-3-319-78381-9_3. Available from: https://link.springer.com/chapter/10.1007/978-3-319-78381-9%7B%5C_%7D3.
97	NING, Fangxiao et al. Efficient tamper-evident logging of distributed systems via concurrent authenticated tree. In: 2017 IEEE 36th International Performance Computing and Communications Conference, IPCCC 2017. [S.l.]: Institute of Electrical and Electronics Engineers Inc., Feb. 2018. P. 1–9. DOI: 10.1109/PCCC.2017.8280476.
98	ZHANG, Yanhua et al. Efficient lattice FIBS for identities in a small universe. In: COMMUNICATIONS in Computer and Information Science. [S.l.]: Springer Verlag, 2018. P. 83–95. DOI: 10.1007/978-981-13-3095-7_7.
99	GALBRAITH, Steven D. **Authenticated key exchange for SIDH**. [S.l.], 2018. Available from: https://www.math.auckland.ac.nz/%7B~%7Dsgal018/AKE.pdf.
100	BOYEN, Xavier; LI, Qinyi. Almost tight multi-instance multi-ciphertext identity-based encryption on lattices. In: LECTURE Notes in Computer Science. [S.l.]: Springer Verlag, 2018. P. 535–553. DOI: 10.1007/978-3-319-93387-0_28.
101	HUANG, Jianye et al. A Black-Box Construction of Strongly Unforgeable Signature Scheme in the Leakage Setting *. **International Journal of Foundations of Computer Science**, World Scientific Publishing Co. Pte Ltd, v. 9, n. 6, p. 761–780, Sept. 2018. DOI: 10.1142/S0129054117400172. Available from: www.worldscientific.com.
102	MOMENG LIU, Yupu Hu. Universally composable oblivious transfer from ideal lattice. **Front. Comput. Sci**, Higher Education Press, v. 13, n. 4, p. 879–906, Aug. 2019. DOI: 10.1007/s11704-018-6507-4. Available from: https://doi.org/10.1007/s11704-018-6507-4.
103	GEONTAE NOH, Ik Rae Jeong. Transitive signature schemes for undirected graphs from lattices. **KSII Transactions on Internet and Information Systems**, v. 13, n. 6, p. 3316–3332, 2019. ISSN 22881468. DOI: 10.3837/tiis.2019.06.030. Available from: http://itiis.org/digital-library/manuscript/file/22144/TIISVol13No6-30.pdf.
104	ZHANDRY, Mark. The Magic of ELFs. **Journal of Cryptology**, v. 32, n. 3, p. 825–866, 2019. ISSN 14321378. DOI: 10.1007/s00145-018-9289-9. Available from: https://link.springer.com/article/10.1007/s00145-018-9289-9.

105	MIRCHANDANI, Anisha. The GDPR-Blockchain Paradox: Exempting Permissioned Blockchains from the GDPR. **Fordham Intellectual Property, Media and Entertainment Law Journal**, v. 29, n. 4, p. 1201, 2019. Available from: https://ir.lawnet.fordham.edu/iplj.
106	NAVID ALAMATI HART MONTGOMERY, Sikhar Patranabis. Symmetric Primitives with Structured Secrets. In: LECTURE Notes in Computer Science. [S.l.]: Springer Verlag, 2019. P. 650–679. DOI: 10.1007/978-3-030-26948-7_23.
107	DÖTTLING, Nico et al. Rate-1 Trapdoor Functions from the Diffie-Hellman Problem. In: LECTURE Notes in Computer Science. [S.l.]: Springer, 2019. P. 585–606. DOI: 10.1007/978-3-030-34618-8_20.
108	KARTHIK NANDAKUMAR NALINI RATHA, Sharathchandra Pankanti. Proving Multimedia Integrity using Sanitizable Signatures Recorded on Blockchain. **dl.acm.org**, ACM, v. 19, p. 10, July 2019. DOI: 10.1145/3335203.3335729. Available from: https://doi.org/10.1145/3335203.3335729.
109	BLAZY, Olivier et al. Post-Quantum UC-Secure Oblivious Transfer in the Standard Model with Adaptive Corruptions. In: PROCEEDINGS of the 14th International Conference on Availability, Reliability and Security - ARES '19. New York, New York, USA: ACM Press, 2019. Available from: https://doi.org/10.1145/3339252.3339280.
110	YOSHIDA, Yusuke et al. Non-Committing Encryption with Quasi-Optimal Ciphertext-Rate Based on the DDH Problem. In: LECTURE Notes in Computer Science. [S.l.: s.n.], 2019. P. 128–158. DOI: 10.1007/978-3-030-34618-8_5. Available from: https://link.springer.com/chapter/10.1007/978-3-030-34618-8%7B%5C_%7D5.
111	GARG, Sanjam et al. New techniques for efficient trapdoor functions and applications. In: LECTURE Notes in Computer Science. [S.l.]: Springer Verlag, 2019. P. 33–63. DOI: 10.1007/978-3-030-17659-4_2.
112	ALAMATI, Navid et al. Minicrypt primitives with algebraic structure and applications. In: SPRINGER. ANNUAL International Conference on the Theory and Applications of Cryptographic Techniques. [S.l.: s.n.], 2019. P. 55–82.
113	DERLER, David; SLAMANIG, Daniel. Key-homomorphic signatures: definitions and applications to multiparty signatures and non-interactive zero-knowledge. **Designs, Codes, and Cryptography**, Springer New York LLC, v. 87, n. 6, p. 1373–1413, June 2019. ISSN 15737586. DOI: 10.1007/s10623-018-0535-9.
114	PARK, Sunoo; SEALFON, Adam. It Wasn't Me!: Repudiability and Claimability of Ring Signatures. In: LECTURE Notes in Computer Science. [S.l.]: Springer Verlag, 2019. P. 159–190. DOI: 10.1007/978-3-030-26954-8_6.
115	KOCH, Jessica et al. **Improvements and New Constructions of Digital Signatures Doktors der Naturwissenschaften**. [S.l.], 2019. Available from: https://pdfs.semanticscholar.org/af9d/8d7d5b548a5afacd0a036e7cf7befc0e57e6.pdf.

116	B, Shuichi Katsumata; YAMADA, Shota. Group Signatures Without NIZK : v. 1, Grant 780701, p. 312–344, 2019. DOI: 10.1007/978-3-030-17659-4. Available from: http://58.194.172.13:80/rwt/154/http/MS6C63DQNEYG86UH/10.1007/978-3-030-17659-4%7B%5C_%7D11.
117	ZHANG, Yanhua et al. Efficient fuzzy identity-based signature from lattices for identities in a small (or large) universe. **Journal of Information Security and Applications**, v. 47, p. 86–93, 2019. ISSN 22142126. DOI: 10.1016/j.jisa.2019.04.012. Available from: https://www.sciencedirect.com/science/article/pii/S2214212618307658.
118	RUFFING, Tim. **Cryptography for Bitcoin and Friends**. [S.l.], 2019. Available from: https://publikationen.sulb.uni-saarland.de/handle/20.500.11880/29102.
119	YU, Bin et al. Chameleon Hash Time-Lock Contract for Privacy Preserving Payment Channel Networks. In: LECTURE Notes in Computer Science. [S.l.]: Springer, 2019. P. 303–318. DOI: 10.1007/978-3-030-31919-9_18. Available from: https://link.springer.com/chapter/10.1007/978-3-030-31919-9%7B%5C_%7D18.
120	AUTHCROPPER: AUTHENTICATED IMAGE CROPPER FOR PRIVACY PRESERVING SURVEILLANCE SYSTEMS. **ACM Trans. Embed. Comput. Syst**, Association for Computing Machinery, v. 18, 5s, Oct. 2019. DOI: 10.1145/3358195. Available from: https://doi.org/10.1145/3358195.
121	YAN, Jianhua et al. Attribute-Based Signcryption from Lattices in the Standard Model. **IEEE Access**, v. 7, p. 56039–56050, 2019. ISSN 21693536. DOI: 10.1109/ACCESS.2019.2900003. Available from: https://ieeexplore.ieee.org/abstract/document/8653267/.
122	DI LUZIO, Adriano et al. Arcula: A Secure Hierarchical Deterministic Wallet for Multi-asset Blockchains, June 2019. arXiv: 1906.05919. Available from: http://arxiv.org/abs/1906.05919.
123	WANG, Xianmin et al. **An Identity-Based Signcryption on Lattice without Trapdoor**. v. 25. [S.l.], 2019. P. 282–293. Available from: http://jucs.org/jucs%7B%5C_%7D25%7B%5C_%7D3/an%7B%5C_%7Didentity%7B%5C_%7Dbased%7B%5C_%7Dsigncryption/jucs%7B%5C_%7D25%7B%5C_%7D03%7B%5C_%7D0282%7B%5C_%7D0293%7B%5C_%7Dwang.pdf.
124	CAO, Nanyuan et al. All-But-Many Lossy Trapdoor Functions under Decisional RSA Subgroup Assumption and Application. **academic.oup.com**, 2019. DOI: 10.1093/comjnl/bxz008. Available from: https://academic.oup.com/comjnl/article/62/8/1148/5369686.
125	THANALAKSHMI, P; ANITHA, R. A Quantum Resistant Chameleon Hashing and Signature Scheme. **IETE Journal of Research**, 2019. ISSN 0974780X. DOI: 10.1080/03772063.2019.1698323. Available from: https://www.tandfonline.com/doi/abs/10.1080/03772063.2019.1698323.
126	ZHANG, Xiao et al. A generic construction of tightly secure signatures in the multi-user setting. **Theoretical Computer Science**, v. 775, p. 32–52, 2019. ISSN 03043975. DOI: 10.1016/j.tcs.2018.12.012. Available from: https://www.sciencedirect.com/science/article/pii/S0304397518307333.

127	CHEN, Liqun et al. A Framework for Efficient Lattice-Based DAA. **dl.acm.org**, Association for Computing Machinery (ACM), p. 23–34, 2019. DOI: 10.1145/3338511.3357349. Available from: https://doi.org/10.1145/3338511.3357349.
128	LU, Xingye et al. (Linkable) Ring signature from hash-then-one-way signature. In: PROCEEDINGS - 2019 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/13th IEEE International Conference on Big Data Science and Engineering, TrustCom/BigDataSE 2019. [S.l.: s.n.], 2019a. P. 578–585. DOI: 10.1109/TrustCom/BigDataSE.2019.00083. Available from: https://ieeexplore.ieee.org/abstract/document/8887320/.
129	DIMA GRIGORIEV, Vladimir Shpilrain. RSA and redactable blockchains, Jan. 2020. arXiv: 2001.10783. Available from: http://arxiv.org/abs/2001.10783.
130	KAI SAMELIN, Daniel Slamanig. Policy-based sanitizable signatures. In: SS20. LECTURE Notes in Computer Science. [S.l.]: Springer, 2020. P. 538–563. DOI: 10.1007/978-3-030-40186-3_23. Available from: https://link.springer.com/chapter/10.1007/978-3-030-40186-3%7B%5C_%7D23.
131	LI, Cong et al. OUP accepted manuscript. **The Computer Journal**, 2020. ISSN 0010-4620. DOI: 10.1093/comjnl/bxaa075. Available from: https://academic.oup.com/comjnl/article-abstract/doi/10.1093/comjnl/bxaa075/5872129.
132	MOJTABA KHALILI MOHAMMAD DAKHILALIAN, Willy Susilo. Efficient chameleon hash functions in the enhanced collision resistant model. **Information Sciences**, v. 510, p. 155–164, 2020. ISSN 00200255. DOI: 10.1016/j.ins.2019.09.001. Available from: https://www.sciencedirect.com/science/article/pii/S002002551930831X.
133	SUBHRA MAZUMDAR, Sushmita Ruj. CryptoMaze: Atomic Off-Chain Payments in Payment Channel Network, May 2020. arXiv: 2005.07574. Available from: http://arxiv.org/abs/2005.07574.
134	DAN MICHAEL A. CORTEZ ARIEL M. SISON, Ruji P. Medina. Cryptographic Randomness Test of the Modified Hashing Function of SHA256 to Address Length Extension Attack. In: ACM International Conference Proceeding Series. [S.l.]: Association for Computing Machinery, Apr. 2020. P. 24–28. DOI: 10.1145/3390525.3390540. Available from: https://doi.org/10.1145/3390525.3390540A.
135	DHARMINDER DHARMINDER, Dheerendra Mishra. Construction of Identity Based Signcryption Using Learning with Rounding. In: COMMUNICATIONS in Computer and Information Science. [S.l.]: Springer, 2020. P. 612–626. DOI: 10.1007/978-981-15-6318-8_49.
136	SUN, Yi et al. An Adaptive Authenticated Data Structure with Privacy-Preserving for Big Data Stream in Cloud. **IEEE Transactions on Information Forensics and Security**, v. 15, p. 3295–3310, 2020. ISSN 15566021. DOI: 10.1109/TIFS.2020.2986879. Available from: https://ieeexplore.ieee.org/abstract/document/9063421/.

137	DARIO CATALANO GEORG FUCHSBAUER, Azam Soleimanian. **Double-authentication-preventing signatures in the standard model**. 12238 LNCS. [S.l.], 2020. P. 338–358. ISBN 9783030579890. DOI: 10 . 1007 / 978 - 3 - 030 - 57990 - 6 _ 17. Available from: https://eprint.iacr.org/2020/789.pdf.

A.5.2 Accepted Results

A.5.2.1 Preimage Chameleon Hash to Increase Signature Security

These papers use the preimage chameleon hash to improve the signature security as described in Subsections 3.4.3 and 4.3.1. In all the cases the authors are using the post-quantum chameleon hash based on lattices to help building post-quantum signature schemes that can be proven in the standard model.

1	CASH, David et al. Bonsai trees, or how to delegate a lattice basis. In: SPRINGER. ANNUAL international conference on the theory and applications of cryptographic techniques. [S.l.: s.n.], 2010. P. 523–552.
2	RÜCKERT, Markus. Adaptively secure identity-based identification from lattices without random oracles. In: SPRINGER. INTERNATIONAL Conference on Security and Cryptography for Networks. [S.l.: s.n.], 2010a. P. 345–362
3	BRAKERSKI, Zvika; KALAI, Yael Tauman. A Framework for Efficient Signatures, Ring Signatures and Identity Based Encryption in the Standard Model. **IACR Cryptol. ePrint Arch.**, v. 2010, p. 86, 2010
4	ZHANDRY, Mark. **Cryptography in the Age of Quantum Computers**. 2015. PhD thesis – Stanford University
5	EATON, Edward. **Signature schemes in the quantum random-oracle model**. 2017. MA thesis – University of Waterloo

A.5.2.2 Preimage Chameleon Hash for Authentication

This thesis use a preimage chameleon hash to instantiate part of a post-quantum key-exchange protocol. The chameleon hash acts as a one-way function with trapdoor.

6	LEGROW, Jason. **Post-quantum security of authenticated key establishment protocols**. 2016. MA thesis – University of Waterloo.

A.5.2.3 Preimage Chameleon Hash for Signature Construction

These are papers which propose using the chameleon hash to construct signature schemes.

7	GORBUNOV, Sergey et al. Leveled fully homomorphic signatures from standard lattices. In: PROCEEDINGS of the forty-seventh annual symposium on Theory of computing. [S.l.: s.n.], 2015. P. 469–477.
8	XIE, Dong et al. Homomorphic signatures from chameleon hash function. **Informational Technology and Control**, v. 46, n. 2, p. 274–286, 2017.
9	LI, BaoHong et al. Lattice-Based Universal Designated Verifier Signatures. In: IEEE. 2018 IEEE 15th Internationalç Conference on e-Business Engineering (ICEBE). [S.l.: s.n.], 2018. P. 329–334.
10	LU, Xingye et al. Raptor: a practical lattice-based (linkable) ring signature. In: SPRINGER. INTERNATIONAL Conference on Applied Cryptography and Network Security. [S.l.: s.n.], 2019c. P. 110–130.

The seventh and eighth work are about homomorphic signature schemes using chameleon hashes, as described in section 5.5.

The ninth work was described in section 5.4.

The tenth work was mentioned in section 5.3. The method proposed to build a ring signature was used in that section to build an unforgeable signature scheme against adaptive chosen message attacks.

A.5.2.4 Quantum Preimage Chameleon Hash

This paper describes a quantum preimage chameleon hash, with some properties that are impossible for classical chameleon hashes. For example, the $CH.\text{PREIMAGE}$ algorithm can be used only once.

11	AMOS, Ryan et al. One-shot signatures and applications to hybrid quantum/classical authentication. In: PROCEEDINGS of the 52nd Annual ACM SIGACT Symposium on Theory of Computing. [S.l.: s.n.], 2020. P. 255–268.

It was not described in this thesis because the preimage chameleon hash described there is a generalization of the scheme studied here. The quantum chameleon hash was outside our scope.

A.5.2.5 Preimage Chameleon Hash to Construct Regular Chameleon Hash

This paper shows how to construct new chameleon hashes from some sigma protocols. One of them is a chameleon hash based on Fiat-Shamir sigma protocol. The construction is similar to the one presented on Subsection 3.3.2, but replacing the one-way function from that construction with a one-way trapdoor function. This makes the construction a preimage chameleon hash.

12	BELLARE, Mihir; RISTOV, Todor. A characterization of chameleon hash functions and new, efficient designs. **Journal of cryptology**, Springer, v. 27, n. 4, p. 799–823, 2014.

A.6 CONCLUSION

The existance of so few results about the concept justify our initial thought that preimages chameleon hashes were a new concept.

Using this systematic review we could answer affirmatively the question about existing previous works dealing with preimage chameleon hash schemes. We also could classify 5 different use cases proposed in literature for preimage chameleon hashes, answering then the second question.

None of the proposed use cases and constructions are equivalent to our propose of building preimage signatures. The proposed homomorphic signature (accepted results 7 and 8) is the most similar, as the chameleon hash of a message and a valid signature will be equal some digest chosen during key creation. However, in that work the digests are chosen uniformly at random and the signature, if not generalized to a tag-based signature as both works propose, can be used to sign a single group of messages for each pair of keys.

The previous constructions also did not guarantee the unforgeability in a security model as rigorous as ours: letting an attacker choose adaptively a polynomially bounded number of signing queries and assuming that the adversary can use quantum algorithms, a scenario where rewinding techniques cannot be used in the proof.