Chameleon Hashes: Definitions and Constructions

Thiago Leucz Astrizi

April 22, 2020

Contents

1 Introduction

Chameleon Hashes are functions similar to common cryptographic hashes. In the chameleon hash, we use an asymmetric cryptographic key pair. The public key is required to determine the message hash. Given a message and the public key, all expected properties, such as collision resistance and hash pre-image, are respected by the chameleon hash.

However, with the private key, the chameleon hash allows, in a controlled way, to determine collisions. Only those who have the private key can determine these collisions.

In this work, we define them and prove some of its properties using the same framework for cryptographic proofs presented by Dan Boneh and Victor Shoup in their forthcoming book [3].

2 Chameleon Hash Schemes

Be $\lambda \geq 1$ a positive integer called **security parameter** and Λ a bit string called **system parameter**.

Definition 1 *A **Simple Chameleon Hash Scheme** CH is a tuple of 3 efficient algorithms $(KeyGen, Hash, Collision)$, along with three families of spaces:*

$$M = \{M_{\lambda,\Lambda}\}_{\lambda,\Lambda}, R = \{R_{\lambda,\Lambda}\}_{\lambda,\Lambda}, C = \{C_{\lambda,\Lambda}\}_{\lambda,\Lambda}$$

Each $S = \{S_{\lambda,\Lambda}\}_{\lambda,\Lambda}$ is a collection of finite sets of bit strings indexed by λ and Λ. For each chameleon hash scheme exists an efficient probabilistic algorithm P that outputs a Λ given λ as input where the lenght of Λ is bounded by a polynomial over λ.

We also have the following properties:

- *All the sets in M, R and C are efficiently recognizable;*
- *All the sets in R are efficiently sampleable;*

- ***KeyGen*** *is an efficient probabilistic algorithm that on input λ, Λ, outputs a pair $(\mathcal{PK}, \mathcal{SK})$ where $\mathcal{PK}$ and $\mathcal{SK}$ are bit strings whose length is bounded by a polynomial in λ. They are respectivelly called* ***public key*** *and* ***private key****. If there's no ambiguity, we ommit the parameters and write just "$(pk, sk) \leftarrow KeyGen()$".*

- ***Hash*** *is an efficient deterministic algorithm that on input λ, Λ, $\mathcal{PK}$, $m \in M_{\lambda,\Lambda}$ and $r \in R_{\lambda,\Lambda}$, outputs an element of $C_{\lambda,\Lambda}$. We call m a* ***message****, r a random parameter and the output a* ***digest****. Sometimes we can ommit the parameters λ, Λ and $\mathcal{PK}$ of this function and write just "$c \leftarrow Hash(m, r)$".*

- ***Collision*** *is an efficient probabilistic algorithm that on input λ, Λ, $\mathcal{SK}$, $m \in M_{\lambda,\Lambda}$, $r \in R_{\lambda,\Lambda}$ and $m' \in M_{\lambda,\Lambda}$, outputs an element $r' \in R_{\lambda,\Lambda}$ such as:*

$$Hash(\lambda, \Lambda, \mathcal{PK}, m, r) = Hash(\lambda, \Lambda, \mathcal{PK}, m', r')$$

If there's no ambiguity, we can ommit the parameters λ, Λ and $\mathcal{SK}$ and write just "$r' \leftarrow Collision(m, r, m')$".

A chameleon hash with these properties were first explicitly proposed in 1997 by Hugo Krawczyk and Tal Rabin. The two authors suggested that they could be used in chameleon signatures. Over the years much more uses were found for them. Chameleon signatures and other applications for chameleon hashes will be discussed later.

Here will also be presented a novel variation for chameleon hashes: the **Preimage Chameleon Hash**:

Definition 2 *A* ***Preimage Chameleon Hash Scheme*** *CH is a tuple of 3 efficient algorithms $(KeyGen, Hash, PreImage)$ defined over three families of spaces (M, R, C) exactly as the Simple Chameleon Hash Scheme. The sets (M, R, C) and the functions $KeyGen, Hash$ have the same properties. But instead of function $Collision$, we have a function $PreImage$ with the following property:*

in

- ***PreImage*** *is an efficient probabilistic algorithm that on input λ, Λ, $\mathcal{SK}$, $m \in M_{\lambda,\Lambda}$ and $c \in C_{\lambda,\Lambda}$, outputs an element r such as:*

$$Hash(\lambda, \Lambda, \mathcal{PK}, m, r) = c$$

If there's no ambiguity, we can ommit the first parameters in the function and write just $r \leftarrow PreImage(m, c)$.

Preimage Chameleon Hashes shouldn't be confused with One-Way Functions with Trapdoors. Their random parameter gives them more flexibility, as we can make any possible message result in any possible digest just selecting the right r.

Theorem 1 *Every Preimage Chameleon Hash is also a Simple Chameleon Hash.*

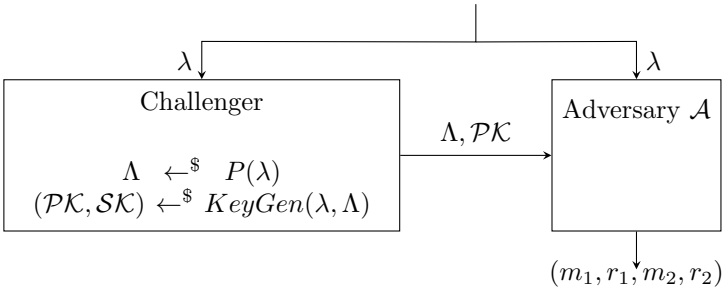


Figure 1: Attack Game 1: Collision Resistance. Adversary $\mathcal{A}$ wins if $(m_1, r_1) \neq (m_2, r_2)$ and $Hash(m_1, r_1) = Hash(m_2, r_2)$.

Proof: We can define a $Collision$ function from a $PreImage$ function:

$$Collision(m, r, m') = PreImage(m', Hash(m, r))$$

□

The inverse isn't true. Preimage Chameleon Hashes are a more powerful version of chameleon hash.

As with regular hash schemes, some properties are very desirable in the two chameleon hash schemes.

Attack Game 1 ***(Collision Resistance).*** *For a given chameleon hash scheme CH defined over (M, R, C), and an adversary $\mathcal{A}$, the adversary takes as input λ, Λ and $\mathcal{PK}$ and outputs $m_1, m_2 \in M_{\lambda,\Lambda}$ and $r_1, r_2 \in R_{\lambda,\Lambda}$.*

We say that $\mathcal{A}$ wins the game if $(m_1, r_1) \neq (m_2, r_2)$ and:

$$Hash(\lambda, \Lambda, \mathcal{PK}, m_1, r_1) = Hash(\lambda, \Lambda, \mathcal{PK}, m_2, r_2)$$

We define $\mathcal{A}$'s advantage with respect to CH as the function with outputs the probability that $\mathcal{A}$ wins the game given λ. We denote the advantage as $Adv_{ColRes}[\mathcal{A}, CH](\lambda)$. We also call $\mathcal{A}$ a ***collision finder****.*

Definition 3 *We call a chameleon hash scheme* ***collision resistent*** *if for all efficient adversaries $\mathcal{A}$, $Adv_{ColRes}[\mathcal{A}, CH](\lambda)$ is a negligible function.*

We notice that a chameleon hash scheme can't be collision resistant if $|C_{\lambda,\Lambda}|$ isn't superpolynomial with respect to λ, as a naive attacker whitch chooses random values of (m_1, r_1, m_2, r_2) succeds with probability $1/|C_{\lambda,\Lambda}|$.

Attack Game 2 ***Preimage Resistance:*** *For a given chameleon hash scheme CH defined over (M, R, C), and an adversary $\mathcal{A}$, a challenger sends to the adversary λ, Λ, $\mathcal{PK}$ and $c \in C_{\lambda,\Lambda}$ and the adversary outputs $m' \in M_{\lambda,\Lambda}$ and $r' \in R_{\lambda,\Lambda}$.*

We say that the adversary $\mathcal{A}$ wins the game if $Hash(\lambda, \Lambda, \mathcal{PK}, m', r') = c$

We define $\mathcal{A}$'s advantage with respect to CH as the function whitch outputs the probability that $\mathcal{A}$ wins game given λ. We denote this advantage as $Adv_{PreImg}[\mathcal{A}, CH](\lambda)$.

Definition 4 *We call a chameleon hash scheme* ***preimage resistent*** *or an* ***one-way function*** *if for all efficient adversaries $\mathcal{A}$, $Adv_{PreImg}[\mathcal{A}, CH](\lambda)$ is a negligible function.*

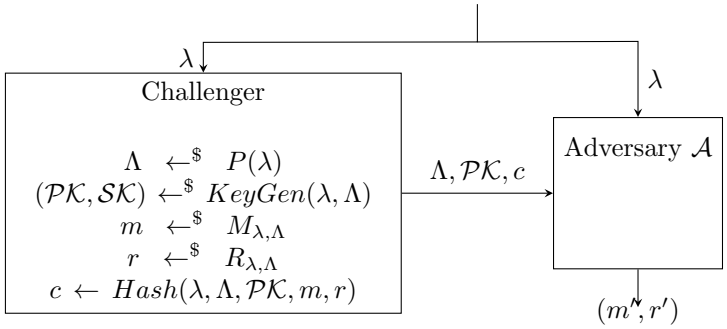


Figure 2: Attack Game 2: Preimage Resistance. Adversary $\mathcal{A}$ wins if $Hash(m', r') = c$.

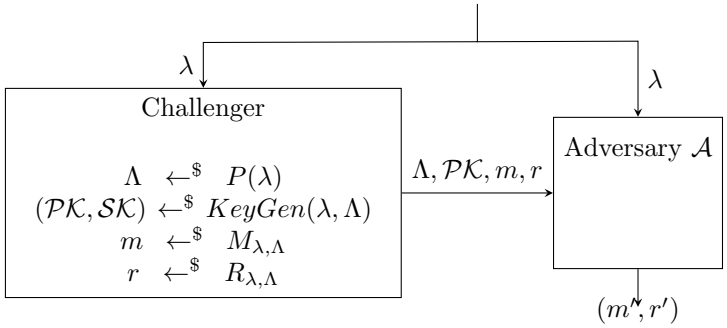


Figure 3: Attack Game 3: Second-Preimage Resistance. Adversary $\mathcal{A}$ wins if $(m, r) \neq (m', r')$ and $Hash(m, r) = Hash(m', r')$.

We can define a related property of **second-preimage resistance** with the following attack game:

Attack Game 3 ***Second-Preimage Resistance:*** *For a given chameleon hash scheme CH defined over (M, R, C), and an adversary $\mathcal{A}$, a challenger sends to the adversary λ, Λ, $\mathcal{PK}$, $m \in M_{\lambda,\Lambda}$ and $r \in R_{\lambda,\Lambda}$ and the adversary outputs $m' \in M_{\lambda,\Lambda}$ and $r' \in R_{\lambda,\Lambda}$.*

We say that the adversary $\mathcal{A}$ wins the game if $Hash(\lambda, \Lambda, \mathcal{PK}, m', r') = Hash(\lambda, \Lambda, \mathcal{PK}, m, r)$ and $(m, r) \neq (m', r')$.

We define $\mathcal{A}$'s advantage with respect to CH as the function whitch outputs the probability that $\mathcal{A}$ wins game given λ. We denote this advantage as $Adv_{2PreImg}[\mathcal{A}, CH](\lambda)$.

Definition 5 *We call a chameleon hash scheme* ***second-preimage resistent*** *if for all efficient adversaries $\mathcal{A}$, $Adv_{2PreImg}[\mathcal{A}, CH](\lambda)$ is a negligible function.*

Unlike in usual hash functions, in chameleon hashes, all pairs of possible input (m, r) have a second-preimage. The existance of functions $Collision$ and $PreImage$ guarantees this. From $Collision$ and $PreImage$ function definitions, given any pair (m, r), we can find at least $|M_{\lambda,\Lambda}| - 1$ second-preimages.

This shows that the number of different input for our $Hash$ function is at least the number of possible digests multiplied by the number of possible

messages: $|M_{\lambda,\Lambda}|.|R_{\lambda,\Lambda}| \geq |M_{\lambda,\Lambda}|.|C_{\lambda,\Lambda}|$. So the number of different random parameters must always be at least the same as the number of different digests.

Theorem 2 *If a chameleon hash scheme is collision resistant, then it's also second-preimage resistant.*

Proof: We will prove that if the chameleon hash is not second-preimage resistant, then it's also not collision resistant.

If exists some efficient adversary $\mathcal{A}$ which can win the attack game 3 with non-negligible probability, then we can build an efficient adversary $\mathcal{B}$ which can win the Attack Game 1 with non-negligible probability.

The adversary $\mathcal{B}$ gets λ, Λ and $\mathcal{PK}$ as input. It can then generate $m \leftarrow^{\$} M_{\lambda,\Lambda}$ and $r \leftarrow^{\$} R_{\lambda,\Lambda}$ and pass all these values as input to $\mathcal{A}$. I gets (m', r') from $\mathcal{A}$'s output and returns (m, r, m', r').

We have:

$$Adv_{ColRes}[B, CH](\lambda) = Adv_{2PreImg}[A, CH](\lambda)$$

So exists such efficient $\mathcal{B}$ given that $\mathcal{A}$ exists. □

Theorem 3 *If for a chameleon hash scheme is second-preimage resistant, then it's also preimage resistant.*

Proof: We will show that if we find an adversary $\mathcal{A}$ which could win the Attack Game 2 with non-negligible probability, we could build an adversary $\mathcal{B}$ which uses $\mathcal{A}$ to win the attack game for second-preimage resistance.

Our adversary $\mathcal{B}$ just get λ and $(\Lambda, \mathcal{PK}, m, r)$ as input and then computes $c = Hash(\lambda, \Lambda, \mathcal{PK}, m, r)$. It then passes λ and $(\Lambda, \mathcal{PK}, c)$ to adversary $\mathcal{A}$. Then it collects and return as output $\mathcal{A}$'s output.

As all the pairs (m, r) always have at least $|M_{\lambda,\Lambda}|$ preimages, if $\mathcal{A}$ returns a correct output, the only case when $\mathcal{B}$ wouldn't win the game would be if $(m, r) = (m', r')$. This happens with probability $1/|M_{\lambda,\Lambda}|$ in the worst case. So:

$$\left(1 - \frac{1}{|M_{\lambda,\Lambda}|}\right) Adv_{PreImg}[\mathcal{A}, CH](\lambda) \leq Adv_{2PreImg}[\mathcal{B}, CH](\lambda)$$

So we conclude that if $\mathcal{A}$ wins with non-negligible probability, then $\mathcal{B}$ would also win with non-negligible probability. □

The previous theorem isn't necessarily true for traditional cryptographic hash functions, mainly because for them we can't guarantee that all possible inputs have a second-preimage.

We also need the property that a digest doesn't reveal any information about the message. We define this with the semantic security property.

Attack Game 4 ***Semantic Security:*** *For a given chameleon hash scheme CH defined over (M, R, C), and for a given adversary $\mathcal{A}$, we define two experiments: Experiment 0 and Experiment 1. For $b = 0, 1$, in Experiment b the adversary computes $m_0, m_1 \in M_{\lambda,\Lambda}$ and sends them to the challenger. The challenger chooses uniformly a $r \in R_{\lambda,\Lambda}$ and sends $Hash(m_b, r)$ to the adversary. The adversary outputs a bit $\hat{b} \in \{0, 1\}$.*

For $b = 0, 1$, let W_b be the event that the adversary outputs 1 in experiment b. We define $\mathcal{A}$'s semantic security advantage with respect to CH as:

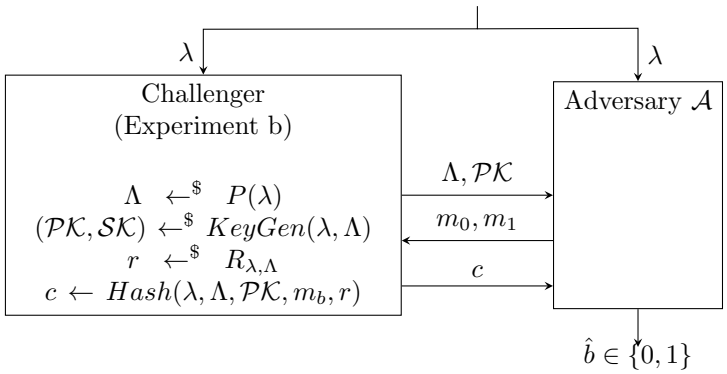


Figure 4: Attack Game 4: Semantic Security. The adversary $\mathcal{A}$ wins if it outputs different values in Experiment 0 ($b = 0$) and Experiment 1 ($b = 1$).

$$Adv_{SS}[\mathcal{A}, CH](\lambda) = |Pr[W_0] - Pr[W_1]|$$

Definition 6 *We call a chameleon hash scheme* ***semantically secure*** *if for all efficient adversaries $\mathcal{A}$, $Adv_{SS}[\mathcal{A}, CH](\lambda)$ is a negligible function.*

For chameleon hashes, any possible message can produce any possible digest, given the correct random parameter. But the semantic security ensures that some digests aren't more probable for some messages than for others. Otherwise an adversary could identify if a random parameter was chosen randomly or was produced with a $Collision$ or $PreImage$ function. We could assume that an improbable digest for a given message was created with a parameter returned by $Collision$ or $PreImage$, not chosen randomly.

The previous security definitions are necessary, but still insufficient to model realistic scenarios of chameleon hash usage. Unlike regular hash functions, here we have the algorithms $Collision$ and $PreImage$ which could be used to create publicly known examples of collisions. An attacker could also induce the owner of a private key to create a desired collision. To represent these requirements, the security definitions must be strengthened allowing an adversary to query for examples of collisions or preimages.

All the previous definitions can be defined under this stronger attack model. This is analogous as how encription can be defined under the chosen plaintext model, where an attacker can query the challenger for examples of encription. But here, as we are dealing with collisions of hash messages, we will call this attack model the **chosen collision attack**.

Definition 7 *A security property defined for a chameleon hash CH as the advantage of adversaries under an attack game is said under* ***chosen collision attack*** *if in its attack game, before returning the final output, the adversary can make Q queries in the following formats:*

- *For query i, the Adversary sends (m_i, r_i, m'_i) with $m_i, m'i \in M_{\lambda,\Lambda}$ and $r_i \in R_{\lambda,\Lambda}$. The Challenger reply with r'_i such as $r_i = Collision(\lambda, \Lambda, \mathcal{SK}, m_i, r_i, m'_i)$.*

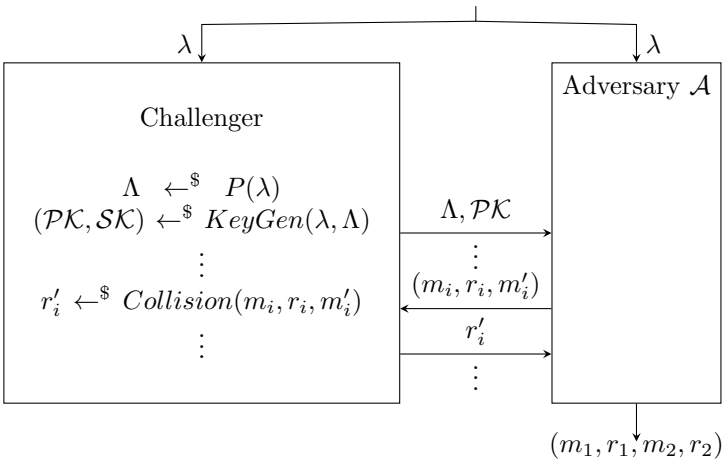


Figure 5: Example of Attack Game under Chosen Collision Attack: Collision Resistance under Chosen Collision Attack. The adversary $\mathcal{A}$ wins if after making Q queries (m_i, r_i, m'_i) with $i \in \{1, \ldots, Q\}$, it outputs (m_1, r_1, m_2, r_2) such as $(m_1, r_1) \neq (m_2, r_2)$, $(m_2, r_2) \neq (m_i, r_i)$ and $(m_2, r_2) \neq (m'_i, r'_i)$ for all $i \in \{1, \ldots, Q\}$.

- *For query i, the Adversary sends (m_i, c_i) and the Challenger reply with $r_i = PreImage(\lambda, \Lambda, \mathcal{SK}, m_i, c_i)$ if CH is a Preimage Chameleon Hash, or reply with "**unknown**" otherwise.*

- *If the property already required the adversary to query values (like in the Semantic Security Attack Game, when the adversary sends m_1, m_2), now the adversary can also repeat this query with other values (in Semantic Security under Chosen Collision the adversary can send queries (m_{0i}, m_{1i}) and get as answer c_i such as $c_i = Hash(m_{bi}, r)$).*

Such queries are adaptative, the Adversary can choose each one after analyzing the previous queries result.

After the queries, if the adversary outputs any pair (m, r), for $m \in M_{\lambda,\Lambda}$ or $R_{\lambda,\Lambda}$, at least one of these pairs (m, r) must have the property that $(m, r) \neq (m_i, r_i)$ and $(m, r) \neq (m'_i, r'_i)$ for all $i \in \{1, \ldots, Q\}$.

If an adversary $\mathcal{A}$ have some advantage $P_{ADV}[\mathcal{A}, CH]$ in a weaker attack model, we say that it has the advantage $P^{CC}_{ADV}[\mathcal{A}, CH]$ under the chosen collision attack.

As an example, Figure 7 shows the Attack Game of Collision Resistance under Chosen Collision Attack.

This is our strongest attack model. It's easy to see that any chameleon hash collision resistant under chosen collision attack is also collision resistant under the weaker attack model. The converse isn't true. If you reveal for an adversary that $r' = Collision(m, r, m')$, this leaks information. In the worst case, the adversary can use this to extract the challenger's private key. If this happens, the chameleon hash isn't collision resistant nor second-preimage resistant under chosen collision attack, even if it has these properties in the weaker model.

Not always this strong attack model is ideal. In some chameleon hash applications, it's desired to be able to compute collisions even without the private key if we have previous examples of collisions which result in the same digest. This happens in Chameleon Signatures Schemes. Some important chameleon hash constructions can also be insecure under chosen collision attack, but secure under new and intermediate attack models. If necessary, we define other attack models when presenting these applications and constructions.

Theorem 4 *If for a chameleon hash scheme CH, if $|R_{\lambda,\Lambda}|$ is superpolynomial and for all tuples (c, m_1, m_2) with $c \in C_{\lambda,\Lambda}$ and $m_1, m_2 \in M_{\lambda,\Lambda}$, the number of random parameters $r_1 \in R_1$ such as $Hash(m_1, r_1) = c$ and $r_2 \in R_2$ such as $Hash(m_2, r_2) = c$, if $|R_1| \approx |R_2|$, then CH is semantically secure and is also semantically secureunder chosen collision attack.*

Proof: TODO. □

3 Construção Baseada em Permutações de Resíduos Quadráticos (Krawczyk) (2000) [5]

3.1 Funções

KeyGen: Como chave pública, gere duas funções de permutação que atuam sobre um mesmo Domínio. Chamaremos elas de f_0 e f_1. Como chave privada, armazene o inverso destas funções: f_0^{-1} e f_1^{-1}.

A construção de Krawczyk usa como funções: $f_0(x) = x^2 \mod n$ e $f_1(x) = 4x^2 \mod n$ para n sendo um múltiplo de dois primos grandes p e q, tal que $p \equiv 3 \mod 8$ e $q \equiv 7 \mod 8$. E sendo a função definida apenas para resíduos quadráticos módulo n que também são primos em relação à n. A chave pública precisa então apenas armazenar n e a chave privada armazena p e q. As funções inversas envolvem raíz quadrada modular, as quais só sabemos como resolver eficientemente módulo n quando conhecemos os fatores de n.

Hash: A função aceita como parâmetro aleatório somente valores de r que são resíduos quadráticos módulo n. Em seguida, usando a representação binária da mensagem m, percorremos em ordem cada um dos bits em uma iteração. Inicialmente o valor do hash é considerado como igual a r. Em cada bit da iteração atualizamos como novo valor da hash o valor anterior passado para f_0 se o bit for 0 ou o valor anterior passado para f_1 se o bit for 1.

PreImage: Para obter um valor válido de r para que a mensagem m tenha o hash C, comece com o valor inicial de C e percorra a representação binária de m de trás pra frente. Toda vez que encontrar um 0, aplique sobre o valor atual de r a função f_0^{-1}. Sempre que encontrar um 1, aplique f_1^{-1}. O valor final é o r desejado.

Collision: Basta escolher uma mensagem m' qualquer que seja diferente de m e calcular usando a equação:

$Collision(\mathcal{SK}, m, r) = (m', PreImage(\mathcal{SK}, m', Hash(\mathcal{PK}, m, r)))$

Isso funciona pelo fato de que toda mensagem neste esquema é capaz de produzir qualquer $c \in C$, bastando que se mude o valor de r utilizado no hash.

3.2 A Exposição de Chave

Se encontramos uma colisão nesta hash camaleão, temos duas mensagens que à partir de algum momento em que iteramos sobre seus bits, obtemos valores distintos x e y tais que:

$$x^2 \equiv 4y^2 \pmod{n}$$

O que nos leva a:

$$x^2 - 4y^2 \equiv 0 \pmod{n}$$

$$(x + 2y)(x - 2y) \equiv 0 \pmod{n}$$

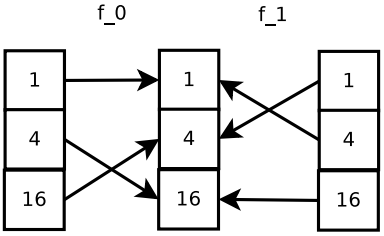
Desta forma, conseguimos fatorar n e ter acesso à chave privada $\mathcal{SK}$.

O mesmo raciocínio também nos mostra que encontrar uma colisão sem acesso à $\mathcal{SK}$ é tão difícil quanto fatorar números.

3.3 Exemplo Didático

Começamos escolhendo os valores $p = 3$ e $q = 7$, que tem a propriedade desejada de serem respectivamente côngruos a 3 e 7 módulo 8. Multiplicamos estes números para obtermos $n = 21 = pq$. No módulo 21 obtemos 3 números que são resíduos quadráticos: 1, 4 e 16.

Com estes três valores diferentes, definimos a função $f_0(x) = x^2 \mod 21$ e $f_1 = 4x^2 \mod 21$:



Para gerarmos o hash da mensagem com bits 01101, geramos um r aleatório que pode ser 1, 4 ou 16. No caso, escolhemos 4. Calculamos então:

$$f_1(f_0(f_1(f_1(f_0(4))))) = f_1(f_0(f_1(f_1(16)))) = f_1(f_0(f_1(16))) = f_1(f_0(16)) = f_1(4) = 1$$

Queremos que a mensagem com bits 00101 tenha exatamente o mesmo hash que a mensagem acima (no caso, 1). Para descobrirmos o valor de r' que devemos associar à ela para este fim, calculamos:

$$f_0^{-1}(f_0^{-1}(f_1^{-1}(f_0^{-1}(f_1^{-1}(1))))) = f_0^{-1}(f_0^{-1}(f_1^{-1}(f_0^{-1}(4)))) = f_0^{-1}(f_0^{-1}(f_1^{-1}(16))) =$$
$$= f_0^{-1}(f_0^{-1}(16)) = f_0^{-1}(4) = 16$$

O que significa que podemos calcular o hash dessa mensagem e obteremos $Hash(00101, 16) = 1$. O que podemos conferir calculando abaixo:

$$f_1(f_0(f_1(f_0(f_0(16))))) = f_1(f_0(f_1(f_0(4)))) = f_1(f_0(f_1(16))) = f_1(f_0(16)) = f_1(4) = 1$$

Exatamente o valor que era esperado.

3.4 Medida de Desempenho

Usando uma implementação em C baseada na biblioteca GNU MP com chaves de 2048 bits e mensagens de 54 bytes, rodando em um computador Intel i7, foi obtido os seguintes tempos de execução:

KeyGen	0,43400s	Hash	0,00104s	Collision	1,79000s

4 Construção Baseada em Logaritmo Discreto (Krawczyk) (2000) [5]

4.1 Funções

KeyGen: Primeiro encontre primos grandes p e q tais que $p = kq + 1$ e um grupo multiplicativo modular $\mathbb{Z}_p^*$ com um gerador g.

A chave privada é um valor aleatório $x \in \mathbb{Z}_q^*$.

A chave pública é um valor $y = g^x \mod p$.

Hash: Dada uma mensagem $m \in \mathbb{Z}_q^*$ e um parâmetro $r \in \mathbb{Z}_q^*$, a hash camaleão é obtida por meio da definição:

$$Hash(\mathcal{PK} = y, m, r) = g^m y^r \mod p$$

PreImage: Não. Seria necessário resolver o problema do logaritmo discreto para implementar esta função.

Collision: A chave privada é o logaritmo discreto de y. Se temos um m e um r que possui um hash conhecido e queremos que um m' qualquer gere o mesmo valor hash, o valor r' necessário é retornado por:

$$Collision(\mathcal{SK} = x, m, r, m') = (m + xr - m')(x)^{-1}$$

Este esquema não é livre de exposição de chaves. A chave secreta x pode ser obtida por qualquer um que tenha acesso à uma colisão (m, r) e (m', r') por meio da fórmula $m + xr = m' + xr'$.

4.2 Exemplo Didático

Vamos escolher um $p = 11 = (5)(2) + 1$ com um $q = 5$. Para o nosso valor de p temos então o seguinte grupo multiplicativo:

×	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	1	3	5	7	9
3	3	6	9	1	4	7	10	2	5	8
4	4	8	1	5	9	2	6	10	3	7
5	5	10	4	9	3	8	2	7	1	6
6	6	1	7	2	8	3	9	4	10	5
7	7	3	10	6	2	9	5	1	8	4
8	8	5	2	10	7	4	1	9	6	3
9	9	7	5	3	1	10	8	6	4	2
10	10	9	8	7	6	5	4	3	2	1

Com relação à exponenciação, se fizermos o valor de cada linha abaixo elevado ao valor da coluna obtemos a tabela:

a^b	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	5	10	9	7	3	6	1
3	3	9	5	4	1	3	9	5	4	1
4	4	5	9	3	1	4	5	9	3	1
5	5	3	4	9	1	5	3	4	9	1
6	6	3	7	9	10	5	8	4	2	1
7	7	5	2	3	10	4	6	9	8	1
8	8	9	6	4	10	3	2	5	7	1
9	9	4	3	5	1	9	4	3	5	1
10	10	1	10	1	10	1	10	1	10	1

Podemos escolher qualquer elemento cuja ordem é $q = 5$ como gerador. Pode ser o 3, 4, 5 ou 9. Escolheremos então $g = 5$. Podemos escolher como chave privada os valores 1, 2, 3 ou 4, pois eles tem ordem $q = 5$. Façamos então $\mathcal{SK} = 3$. O que faz com que nossa chave pública seja $\mathcal{PK} = 5^3 \mod 11 = 4$.

Vamos calcular um hash para a mensagem representada pelo número $m = 8$. O nosso parâmetro aleatório r pode ser 1, 2, 3 ou 4. Façamos então $r = 2$. A hash é calculada então por:

$$Hash(\mathcal{PK} = 4, m = 8, r = 2) = g^m y^r = 5^8 4^2 \mod 11 = 9$$

Agora supondo que tenhamos a chave privada, queremos fazer com que a mensagem $m' = 7$ tenha exatamente o mesmo valor de hash. Para isso fazemos:

$$\begin{aligned} UForge(\mathcal{SK} = 3, m = 8, r = 2, m' = 7) &= \frac{m + xr - m'}{x} \mod 5 \\ &= 2(3)^{-1} \mod 5 = (2)(2) \mod 5 = 4 \end{aligned} \tag{1}$$

Para checar que isso é correto, vamos calcular:

$$Hash(\mathcal{PK} = 4, m' = 7, r' = 4) = g^{m'} y^{r'} = 5^7 4^4 \mod 11 = (3)(3) = 9$$

Exatamente o resultado esperado.

4.3 Medida de Desempenho

KeyGen	0,005490s	Hash	0,009530s	Collision	0,000032s

5 Construção Baseada em Assinaturas Nyberg-Rueppel (Ateniese e Medeiros) (2005) [1]

Esta construção se baseia na existência do esquema de assinaturas Nyberg-Rueppel, um dos esquemas baseados no ElGamal. Nele, começamos gerando dois números primos p e q tais que $p = 2q + 1$. Escolhemos também um gerador g do subgrupo de resíduos quadráticos de $\mathbb{Z}_p$ de ordem q. Escolhemos como

chave privada um número x módulo q e escolhemos como chave pública um valor $y = g^x \mod p$.

Em uma das variações dessa assinatura, assinar uma mensagem m é feito por meio da equação:

$$Sign(m) = (r, s) = ((g^k \mod p) \mod q, k - rmx \mod q)$$

E uma assinatura (r, s) é válida para uma mensagem m se:

$$r = (g^s y^{rm} \mod p) \mod q$$

O valor k acima deve ser sempre escolhido aleatoriamente e não pode ser revelado, ou a chave privada poderia ser facilmente extraída.

A ideia será gerar um esquema de hash camaleão no qual a equação de verificação acima será usada para o nosso hash e a equação de assinatura será usada para obter uma pré-imagem após fazermos algumas modificações. A única mudança que precisará ser feita será a substituição do produto mr pelo resultado do hash convencional $\mathcal{H}(m||r)$ em todas as equações acima. Desta forma, ficaremos com as equações abaixo que servirão de base para nosso hash camaleão.

Equações de verificação:

$$r = (g^k \mod p) \mod q$$

$$s = k - \mathcal{H}(m||r)x \mod q$$

Equação de assinatura:

$$r = (g^s y^{\mathcal{H}(m||r)} \mod p) \mod q$$

A Compreender: Por que foi necessário fazer a substituição de mr por $\mathcal{H}(m||r)$? A rigor, o esquema continuaria funcionando se essa substituição não tivesse sido feita.

5.1 Funções

KeyGen: A geração de chaves é idêntica à do esquema de assinatura. Começamos gerando dois números primos p e q tais que $p = 2q + 1$. Escolhemos também um gerador g do subgrupo de resíduos quadráticos de $\mathbb{Z}_p$ de ordem q. Escolhemos como chave privada um número x módulo q e escolhemos como chave pública um valor $y = g^x \mod p$.

Hash: Geraremos uma função de hash que será baseada na verificação para saber se uma assinatura Nyberg-Rueppel é válida para uma mensagem m. Nosso hash será definido como:

$$Hash(m, (r, s)) = r - (g^s y^{\mathcal{H}(m,r)} \mod p) \mod q$$

A função hash pode então ser interpretada intuitivamente como uma forma de calcular a diferença entre o parâmetro aleatório (r, s) e uma assinatura legítima de Nyberg-Rueppel. Se por uma grande coincidência escolhêssemos como valores (r, s) uma assinatura Nyberg-Rueppel legítima da mensagem m,

então isso significa que o valor de nosso hash seria zero. Nos demais casos, o hash assume qualquer outro valor módulo q.

PreImage: Se a nossa função hash pode ser vista intuitivamente como a diferença entre o parâmetro aleatório (r, s) e uma assinatura legítima, então a nossa função de pré-imagem pode ser vista como um método que calcula uma assinatura legítima Nyberg-Rueppel e depois adiciona uma perturbação na assinatura para que ela fique com a diferença desejada em relação à uma assinatura legítima.

Dada uma mensagem m e o *digest* desejado c, podemos gerar um valor (r, s) fazendo:

$$r = c + (g^k \mod p) \mod q$$

$$s = k - \mathcal{H}(m||r)x \mod q$$

Prova: Demonstraremos que $Hash(m, PreImage(c, m)) = c$.

$$\begin{aligned}
&Hash(m, PreImage(c, m)) \\
&= Hash(m, (c + (g^k \mod p) \mod q, k - x\mathcal{H}(m, r) \mod q)) \\
&= r - (g^s y^{\mathcal{H}(m,r)} \mod p) \mod q \\
&= r - (g^{x\mathcal{H}(m,r)+s} \mod p) \mod q \\
&= r - (g^{x\mathcal{H}(m,r)+k-x\mathcal{H}(m,r)} \mod p) \mod q \\
&= c + (g^k \mod p) - (g^k \mod p) \mod q \\
&= c
\end{aligned}$$

Collision: Como existe uma função $PreImage$, podemos obter uma colisão calculando:

$$Collision(\mathcal{S}\|, m, r, m') = PreImage(\mathcal{SK}, Hash(\mathcal{PK}, m, r), m')$$

5.2 Variações

Ateniese e Medeiros, no mesmo artigo em que apresentou esta construção, apontou que é possível definir a função hash como:

$$Hash(\mathcal{PK}, m, (r, s)) = \left(ry^{\mathcal{H}(m||r)}g^s\right) \mod p$$

Em tal caso, para obter a função $PreImage$, calculamos:

$$r' = Cg^{-k} \mod p$$

$$s' = k - \mathcal{H}(m||r')x \mod q$$

Pode-se provar que isso também funciona mostrando que $Hash(m, PreImage(m, c)) = c$:

$$
\begin{aligned}
Hash(m, PreImage(m,c)) &= Hash(m, (cg^{-k} \mod p, k - \mathcal{H}(m||r)x \mod q)) \\
&= (cg^{-k} \mod p) y^{\mathcal{H}(m||r)} g^{k-\mathcal{H}(m||r)x \mod q} \mod p \\
&= (cg^{-k} \mod p) g^{\mathcal{H}(m||r)x} g^{k-\mathcal{H}(m||r)x \mod q} \mod p \\
&= cg^{-k+\mathcal{H}(m||r)x+k-\mathcal{H}(m||r)x \mod q} \mod p \\
&= c
\end{aligned}
\tag{2}
$$

O artigo que introduz a assinatura de Nyberg-Rueppel [8] menciona um total de 5 variações de cálculo de assinatura e verificação que podem ser adaptados para esquemas de hash camaleão bastante semelhantes. A imagem abaixo retirada do artigo apresdenta todas as variações:

Scheme	Signature	Recovery / Verification
p-NEW	$r = g^k \bmod p$ $s = k - r'mx \bmod q$	$r = g^s y^{r'm} \bmod p$
MR(p)-NEW	$r = mg^{-k} \bmod p$ $s = k - r'x \bmod q$	$m = g^s y^{r'} r \bmod p$
q-NEW	$r = (g^k \bmod p) \bmod q$ $s = k - rmx \bmod q$	$r = (g^s y^{rm} \bmod p) \bmod q$
MR(q)-NEW	$r = m(g^k \bmod p) \bmod q$ $s = k - rx \bmod q$	$m = (g^s y^r \bmod p)^{-1} r \bmod q$
Reduced MR(p)-NEW	$r = (mg^{-k} \bmod p) \bmod q$ $s = k - rx \bmod q$	$r = (mg^{-s} y^{-r} \bmod p) \bmod q$

Todas as variações acima foram geradas à partir do mesmo conjunto inicial de equações e por isso são relacionados.

O q-NEW foi o primeiro esquema que mostramos e mostra-se o mais atrativo por não precisar de qualquer cálculo de inverso multiplicativo. O p-NEW é quase igual, sendo que a principal diferença é em qual momento calculamos o módulo q do r ($r' = r \mod q$).

O MR(p)-NEW é a variação que vimos logo depois e que também foi apresentada por Ateniese e Medeiros na forma adaptada de um hash camaleão.

Podemos mostrar aqui também como seria um hash camaleão baseado na forma MR(q)-NEW e Reduced MR(p)-NEW.

O hash camaleão MR(q)-NEW seria:

$$Hash(m, (r,s)) = (g^s y^{\mathcal{H}(m||r) \mod p})^{-1} r \mod q$$

$$PreImage(c,m) = (c(g^k \mod p) \mod q, k - \mathcal{H}(m||r)x \mod q)$$

Prova: Demonstraremos que $Hash(m, PreImage(c,m)) = c$.

$$
\begin{aligned}
&Hash(m, PreImage(c, m)) \\
&= Hash(m, (cg^k \mod p, k - x\mathcal{H}(m||r) \mod q)) \\
&= (g^s y^{\mathcal{H}(m||r)})^{-1} r \mod p \\
&= (g^{s+x\mathcal{H}(m||r)})^{-1} r \mod p \\
&= (g^{k-x\mathcal{H}(m||r)+x\mathcal{H}(m||r)})^{-1} r \mod p \\
&= g^{-k} c g^k \mod p \\
&= c
\end{aligned}
$$

Por fim, o hash camaleão Reduced MR(p)-NEW seria:

$$Hash(m, (r, s)) = r - (mg^{-s} y^{-r} \mod p) \mod q$$

$$PreImage(c, m) = (c + (mg^{-k} \mod p) \mod q, k - rx \mod q)$$

Prova: Demonstraremos que $Hash(m, PreImage(c, m)) = c$.

$$
\begin{aligned}
&Hash(m, PreImage(c, m)) \\
&= Hash(m, (c + (mg^{-k} \mod p) \mod q, k - rx \mod q)) \\
&= r - (mg^{-s} y^{-r} \mod p) \mod q \\
&= r - (mg^{-s-xr} \mod p) \mod q \\
&= r - (mg^{-k+rx-xr} \mod p) \mod q \\
&= r - (mg^{-k} \mod p) \mod q \\
&= c + (mg^{-k} \mod p) - (mg^{-k} \mod p) \mod q \\
&= c
\end{aligned}
$$

5.3 Segurança

Uma das vantagens de definir hashes camaleão à partir de assinaturas digitais é que se as assinaturas são seguras, pode-se derivar deste fato muito da segurança das hashes camaleão baseadas nelas.

A resistência à colisão e o fato do esquema ser livre de exposição de chave deriva da própria segurança da assinatura de Nyber Rueppel.

Contudo, não necessariamente um esquema de assinatura continua sendo seguro se uma mesma mensagem é assinada várias vezes e um atacante tem acesso à tais assinaturas. Felizmente, no caso da assinatura de Nyberg-Rueppel, isso foi demonstrado em [7], no Apêndice A.

Já a segurança semântica deste esquema foi demonstrada por Ateniese e Medeiros no artigo que apresentou este esquema de assinatura.

5.4 Medida de Desempenho

Este é o desempenho obtido do hash camaleão de Nyberg-Rueppel q-NEW, apresentado por Ateniese e Medeiros e que mostra-se mais atrativo por não precisar de cálculo de inverso multiplicativo:

KeyGen	0,017544s	Hash	0,019996s	Collision	0,007057s

6 Construção Baseada no DSA

A seção anterior nos leva a questionar se a mesma técnica não poderia ser usada para obter hashes camaleão baseados em outras assinaturas que também seriam derivadas do Elgamal. Seria uma questão de montar um hash à partir da equação de verificação e obter uma função de pré-imagem à partir da equação que define a assinatura.

Esta seção mostra que é possível fazer isso com a assinatura DSA.

6.1 Funções

KeyGen: A geração de chaves funciona de maneira idêntica à geração do DSA. Primeiro escolhe-se dois primos p e q tal que $p-1$ é um múltiplo de q. Escolhemos então um inteiro h aleatório entre 2 e $p-2$. É possível simplesmente usar $h = 2$. Por fim, computamos $g = h^{(p-1)/q} \mod p$. Assim (p, q, g) são os parâmetros de sistema usados pelo algoritmo.

Para a chave privada escolhemos um valor $\mathcal{SK} = x$ menor que q. E a chave pública é $\mathcal{PK} = y = g^x \mod p$;

Hash:

Verificar uma assinatura DSA (r, s) para uma mensagem m envolve checar se é verdade que:

$$(g^{\frac{\mathcal{H}(m)}{s} \mod q} y^{\frac{r}{s} \mod q} \mod p) \mod q = r$$

Baseando-se nessa equação construímos nossa função de hash:

$$Hash(m, (r, s)) = r - (g^{\frac{\mathcal{H}(m)}{s} \mod q} y^{\frac{r}{s} \mod q} \mod p) \mod q$$

PreImage:

Criar uma assinatura legítima DSA para a mensagem m envolve escolher (r, s) tais que:

$$r = (g^k \mod p) \mod q$$

$$s = k^{-1}(\mathcal{H}(m) + xr) \mod q)$$

Exatamente como na primeira assinatura Nyberg-Rueppel da seção anteriror, podemos controlar o valor da hash que definimos aqui apenas perturbando o valor de r somando à ele o valor do *digest* desejado:

$$PreImage(m, c) = (c + (g^k \mod p) \mod q, k^{-1}(\mathcal{H}(m) + xr) \mod q)$$

Prova: Demonstraremos que $Hash(m, PreImage(c, m)) = c$.

$$
\begin{aligned}
&Hash(m, PreImage(c, m)) \\
&= Hash(m, (c + (g^k \mod p) \mod q, k^{-1}(\mathcal{H}(m) + xr) \mod q)) \\
&= r - (g^{\frac{\mathcal{H}(m)}{s} \mod q} y^{\frac{r}{s} \mod q} \mod p) \mod q \\
&= r - (g^{\frac{\mathcal{H}(m)+xr}{s} \mod q} \mod p) \mod q \\
&= r - (g^{\frac{k(\mathcal{H}(m)+xr)}{\mathcal{H}(m)+xr} \mod q} \mod p) \mod q \\
&= c + (g^k \mod p) - (g^k \mod p) \mod q \\
&= c
\end{aligned}
$$

6.2 Segurança

Assim como na seção anterior, a segurança deste esquema deriva da própria segurança do esquema de assinaturas DSA e da propriedade que o DSA continua seguro mesmo que um atacante tenha acesso a mais de uma assinatura válida para uma mesma mensagem. A segunda propriedade foi demonstrada em [7].

A segurança semântica deste esquema ainda está para ser demonstrada.

6.3 Medida de Desempenho

KeyGen	0,0020026s	Hash	0,018312s	Collision	0,006846s

7 Construção Baseada no Protocolo Sigma de Fiat-Shamir (Mihir Bellare e Todor Ristov) (2008) [2]

Esta construção foi obtida em um artigo no qual os autores estavam demonstrando a possibilidade de construir novas funções hash com provas de segurança à partir de protocolos sigma. Eles demonstraram que as funções hash que eles obtiveram também tinham a propriedade de ser hashes camaleão.

7.1 Funções

KeyGen: Gere dois primos aleatórios p e q. Faça $n = pq$.

Como chave pública, gere um vetor s de k números que sejam resíduos quadráticos módulo n, sendo k o número de bits das mensagens a serem recebidas. Para adotar as melhorias propostas no Protocolo Micali-Shamir, tais números devem ser primos pequenos que sejam resíduos quadráticos módulo n, Adicione também à chave o valor n.

Como chave privada, gere um vetor v de k números tal que $v_i = s_i^{-2} \mod n$ e também adicione à chave os fatores p e q.

Hash: Seja a mensagem m um vetor binário com k bits e sendo $r \leq n/2$, a hash é obtida por:

$$Hash(\mathcal{PK}, m, r) = r^2 \prod_{m_i=1} v_i \mod n$$

PreImage: Esta função é definida como:

$$PreImage(\mathcal{SK}, C, m') = \sqrt{C} \prod_{m'_i=1} s_i \mod n$$

A raíz quadrada modular só pode ser calculada por quem conheça os fatores primos de n. A fórmula acima funciona, pois:

$$C = r^2 \prod_{m_i=1} v_i = r^2 \prod_{m_i=1} \frac{1}{s_i^2} \Longrightarrow r = \sqrt{C} \prod_{m'_i=1} s_i$$

7.2 O Problema da Exposição de Chave

Esta construção não é livre de exposição de chave. Se somos capazes de obter uma colisão na qual as duas mensagens que colidem possuem bits 1s nas mesmas posições, para cada bit 1 em comum, podemos gerar uma nova colisão para novas mensagens idênticas, mas que possuem o bit 1 correspondente trocado para 0, mantendo os parâmetros aleatórios das mensagens idênticos.

Sendo assim, a probabilidade de que uma colisão não revele novas colisões derivadas é negligível.

7.3 Medida de Desempenho

KeyGen	4,838347s	Hash	0,000004s	Collision	0,004199s

8 Construção Baseada no Protocolo Sigma de Okamoto (Mihir Bellare e Todor Ristov) (2008) [2]

Esta construção foi obtida assim como a anterior à partir de protocolos sigma já existentes após o autor perceber a relação existente entre protocolos sigma e hashes camaleão. Mas ao contrário do método anterior, esta construção não se destaca por ter vantagem em desempenho comparada à outras construções e nem tem a propriedade desejável de ser livre de exposição de chave.

8.1 Funções

KeyGen: Obtenha um grupo onde calcular o logaritmo discreto é um problema difícil. Escolha como chave pública (g_1, g_2, x) onde g_1 e g_2 são geradores e escolha como chave privada (s_1, s_2) tais que $g_1^{s_1} g_2^{s_2} = x$.

Hash: É definida pela seguinte função:

$$Hash(\mathcal{PK} = (g_1, g_2, x), m, r = (r_1, r_2)) = x^m g_1^{r_1} g_2^{r_2}$$

Collision: Pela propriedade da exponenciação, dados m, m' e $r = (r_1, r_2)$ conhecidos, podemos obter $r' = (r'_1, r'_2)$ tal que $Hash(\mathcal{PK}, m, r) = Hash(\mathcal{PK}, m', r')$ por meio da equação:

$$r'_1 = (m - m')s_1 + r_1$$

$$r_2' = (m - m')s_2 + r_2$$

A mesma equação também nos revela que o esquema não é livre da exposição de chaves, pois diante de uma colisão os valores s_1, s_2 podem ser trivialmente isolados.

Prova: Pode-seprovar o funcionamento desta construção mostrando que $Hash(m', Collision(m, r, m')) = Hash(m, r)$:

$$\begin{aligned} Hash(m', Collision(m, r, m')) &= Hash(m', ((m - m')s_1 + r_1, (m - m')s_2 + r_2)) \\ &= x^{m'} g_1^{(m-m')s_1 + r_1} g_2^{(m-m')s_2 + r_2} \\ &= x^{m'} x^{m-m'} g_1^{r_1} g_2^{r_2} \\ &= Hash(m, r) \end{aligned} \tag{3}$$

Contudo, embora funcione, este esquema não apresenta qualquer vantagem em relação à outros esquemas baseados em logaritmo discreto.

9 Construction using Lattices (David Cash and Daniele Micciancio) (2010 and 2011) [4] [6]

This chameleon hash scheme in fact shows how to construct a chameleon hash from a regular cryptographic hash function H and a probabilistic one-way trapdoor function f.

A probabilistic trapdoor function scheme T is a tuple of algorithms (G, F, F^{-1}) defined over (X, Y). The algorithm G is a probabilistic algorithms that givem the security and system parameters returns a pair of keys $(\mathcal{PK}_T, \mathcal{SK}_T)$, F is a deterministic algorithm that given the security and system parameters, the key $\mathcal{PK}$ and $x \in X$, returns $y \in Y$. And F^{-1} is a probabilistic algorithm that given the security and system parameters, the key $\mathcal{SK}$ and a $y \in Y$, returns a $x \in X$ such as $F(x) = y$.

An adversary $\mathcal{A}$ which tries to break the one-way trapdoor function security interacts with a challenger which computes: $(\mathcal{PK}, \mathcal{SK}) \leftarrow^{\$} G()$, $x \leftarrow^{\$} X$, $y \leftarrow^{\$} F(\mathcal{PK}, x)$. The challenger sends y to the adversary and the adversary outbuts $\hat{x} \in X$. The adversary wins if $F(\mathcal{PK}, \hat{x}) = y$. The one-way trapdoor function is secure if for all efficient adversaries, this probability (represented by OWadv$[\mathcal{A}, T]$) is negligible.

The cryptographic hash function H used here will map elements from set A to set Y. And there's a group defined in the set Y using the + operation.

The Cash's chameleon hash scheme is defined over (A, X, Y).

This is the Cash's chameleon hash scheme key generation algorithm:

$$KeyGen() := G()$$

Given the keys taken from the one-way trapdoor function, we define the $Hash$ algorithm as:

$$Hash(\mathcal{PK}, m, r) := H(m) - F(\mathcal{PK}, r)$$

This is a preimage chameleon hash, as we can define the $PreImage$ algorithm as:

$$PreImage(\mathcal{SK}, m, c) := F^{-1}(\mathcal{SK}, H(m) + c)$$

Theorem 5 *The Cash's chameleon hash scheme defined above is a functional chameleon hash scheme.*

Proof: We will show that the $r \in R$ obtained by the $PreImage$ algorithm results in the digest c when used with the message m. Notice that composing $F(\mathcal{PK}, \cdot)$ and $F^{-1}(\mathcal{SK}, \cdot)$ give us a identity function. The converse isn't true because F^{-1} is a probabilistic algorithm.

$$\begin{aligned} Hash(m, r) &= Hash(m, F^{-1}(\mathcal{SK}, H(m) + c)) \\ &= H(m) - F(\mathcal{PK}, F^{-1}(\mathcal{SK}, H(m) + c)) \\ &= H(m) - H(m) + c) \\ &= c \end{aligned} \tag{4}$$

□

The main problem with this construction is that not necessarily the resulting chameleon hash is secure, even if the hash and the one-way trapdoor function are secure. If we cant't invert the trapdoor function and can't invert the hash function, we could still compute (m, r) and (m', r') such as $H(m) - F(\mathcal{PK}, r) = H(m') - F(\mathcal{PK}, r')$.

When this was originally proposed, wasn't in this general form. The author suggested using this construction with specific primitives: the Ajtai Hash Function and a one-way trapdoor function based in lattices.

9.1 Ajtai's Hash Function

Given a matrix A_1 of integers module q such as $A_1 \in \mathbb{Z}_q^{m \times n}$, we define the Ajtai hash function defined over messages $\overrightarrow{m} \in \{0, 1\}^n$ and digests $t \in \mathbb{Z}_q^m$ as:

$$H_A(\overrightarrow{m}) = A\overrightarrow{m} \mod q$$

To understand if this is a secure hash function, let's check what it means if we can compute the inverse of H_A.

For $H_A(\overrightarrow{x}) = 0$. We know that $A\,\overrightarrow{0} = 0$. And because $A\,\overrightarrow{a} + A\,\overrightarrow{b} = A(\overrightarrow{a} + \overrightarrow{b})$, we know that if two vectors are solutions for the equation $H_A(\overrightarrow{x}) = 0$, it's sum also is a solution. It means that the set of all solutions for this equation is in a lattice:

Definition 8 *A **lattice** is a set of points in n-dimentional space with a periodic structure. Given n linearlly independent vectors $\overrightarrow{b_1}, \ldots \overrightarrow{b_n} \in \mathbb{R}^n$ the lattice generated by them is the set of vectors:*

$$\mathcal{L}(\overrightarrow{b_1}, \ldots \overrightarrow{b_n}) = \left\{ \sum_{i=1}^{n} x_i \overrightarrow{b_i} : x_i \in \mathbb{Z} \right\}$$

*The vectors $\overrightarrow{b_1}, \ldots \overrightarrow{b_n}$ are the **basis** for the lattice.*

But not all the vectors in the lattice are real solutions for the equation. By definition, lattices have infinite vectors, but our number of solutions are clearly finite. As we defined our set of messages as $\{0,1\}^n$, only short vectors in the lattice are proper solutions.

But what if we take the equation $H_A(\overrightarrow{x}) = \overrightarrow{t}$ for some $\overrightarrow{t} \neq \overrightarrow{0}$? In this case, the vector $\overrightarrow{0}$ not necessarily is a solution. But we still have the property that if two vectors are solutions, their sum also is a solution. In this case, the solution isn't necessarily in a lattice, but they are in what is called an **ideal lattice**. We can convert all ideal lattices to an equivalent lattice summing all their vectors with a constant in a way that moves one of its points to the origin.

The problem is that finding short vectors in lattices is believed to be a very difficult problem. Using known methods, we could compute a basis for the lattice or ideal lattice, but not a basis with short vectors. And so, according with the following assumption, we can't do this except with negligible probability:

Assumption 1 ***Short Integer Solution Assumption (SIS$_{n,m,q,\beta}$):*** *Given $A \in \mathbb{Z}_q^{n \times m}$ with entries that consists in m uniformly random vectors as its columns, find a nonzero vector $\overrightarrow{x} \in \mathbb{Z}^n$ such that:*

- $||\overrightarrow{x}|| \leq \beta < q$
- $A\overrightarrow{x} = \overrightarrow{0} \in \mathbb{Z}_q^n$
- $\beta \geq \sqrt{n \log q}$
- $m \geq n \log q$

To ensure the existance of nontrivial solutions, is required:

- $\beta \geq \sqrt{n \log q}$
- $m \geq n \log q$

The probability of finding solutions in this problem for any m, q, β is a negligible function over n.

Given this assumption, we can formulate the following theorem:

Theorem 6 *If the SIS assumption is true, then Ajtai's hash function is a collision resistant hash function.*

Proof: For any adversary $\mathcal{A}$ which tries to compute collisions in Ajtai's hash function, we could create a new adversary $\mathcal{B}$ which tries to find hort vectors in a lattice.

The adversary $\mathcal{B}$ would take the matrix A and use it to define an Ajtai's hash function. It would use $\mathcal{A}$ to get a collision: two values $\overrightarrow{m_1}$ and $\overrightarrow{m_2}$ such that $A\overrightarrow{m_1} = A\overrightarrow{m_2}$.

We know that $A(\overrightarrow{m_1} - \overrightarrow{m_2}) = A\overrightarrow{{}_m 1} - A\overrightarrow{m_2} = 0$, so the adversary $\mathcal{B}$ can compute $\overrightarrow{m_1} - \overrightarrow{m_2}$ as the answer.

As $\overrightarrow{m_1}, \overrightarrow{m_2} \in \{0,1\}^n \subset \mathbb{Z}_q^n$, then $||\overrightarrow{{}_m 1} - A\overrightarrow{m_2}|| \leq \sqrt{n} < sqrtn \log q \leq \beta$. The resulting vector is considered a short vector in the SIS assumption.

The probability of this adversary $\mathcal{B}$ returning a correct answer is as least as high as the probability of $\mathcal{A}$ returning a collision. So:

$$\text{CRadv}[\mathcal{A}, H_A] \leq Pr[\text{Finding a SIS solution}]$$

As the prbability of solving a SIS problem is negligible by our hypothesis, then the Ajtai's hash function H_A is collision resistant. □

To ensure that H_A compresses it's input, we neet do set $|\{0,1\}^m| > |\mathbb{Z}_q^n|$. This condition is always true if $m > n \log q$ as required to ensure security under the SIS assumption.

If we also want a guarantee that H_A is an one-way function, we need to ensure that the number of digests without a second-preimage is negligible. This can be done setting m as $n \log q$ multiplied by some function of n non-constant and greater than one. For example, setting $m = n \log n \log q$.

9.2 Micciancio One-Way Trapdoor Function

The one-way trapdoor function of this scheme is very similar to the previous hash function. The F algorithm in the scheme gets a $\mathcal{PK} = A \in \mathbb{Z}_q^{m \times n}$ and a $\overrightarrow{x} \in \mathbb{Z}^n$ and $\overrightarrow{x}$ has an euclidean norm lesser than some small β and returns a y such that:

$$F(A, \overrightarrow{x}) = y = A\overrightarrow{x} \mod q$$

This is essentially the same computation that in Ajtai's hash function. The main difference is that the matrix A here isn't random and uniform. The majority of columns are, but the last columns hide a trapdoor.

To understand the trapdoor, notice that while it's difficult to invert the function $F(A, \cdot)$ for a random and uniform A, exist some matrices for which invert the function is trivial.

Suppose we have matrix $G \in \mathbb{Z}_q^{n \times w}$ for which is easy to compute $F(A, \cdot)$ and its inverse. Given G, we can construct a family of matrices which could also be inverted reducing their inversion problem to the inversion of G. For example, if we have:

$$A \begin{bmatrix} R \\ I \end{bmatrix} = G$$

for $G \in \mathbb{Z}_q^{n \times w}$, a matrix $A \in \mathbb{Z}^{n \times m}$, the identity matrix $I \in \mathbb{Z}^{w \times w}$ and a trapdoor matrix $R \in \mathbb{Z}_q^{(m-w) \times w}$. We can compute the invrse of $F(A, \overrightarrow{x})$ using this algorithm if we know the trapdoor R and the inverse of $F(G, \cdot)$:

$$\begin{aligned} F^{-1}(\mathcal{SK} = (A, R), \overrightarrow{y}) := & \overrightarrow{p} \leftarrow^{\$} \mathbb{Z}_q^m \\ & \overrightarrow{v} \leftarrow \overrightarrow{y} - A\overrightarrow{p} \mod q \\ & \overrightarrow{z} \leftarrow \text{vector such that } F(G, \overrightarrow{z}) = \overrightarrow{v} \\ & \textbf{return } \overrightarrow{p} + \begin{bmatrix} R \\ I \end{bmatrix} \overrightarrow{z} \end{aligned}$$

To prove that this works, we just take the value returned by F^{-1} above, and show that the function $F(A, \cdot)$ maps it to $\overrightarrow{y}$ as expected:

$$F(A, \overrightarrow{p} + \begin{bmatrix} R \\ I \end{bmatrix} \overrightarrow{z}) = A(\overrightarrow{p} + \begin{bmatrix} R \\ I \end{bmatrix} \overrightarrow{z})$$
$$= A\overrightarrow{p} + A \begin{bmatrix} R \\ I \end{bmatrix} \overrightarrow{z}$$
$$= A\overrightarrow{p} + G\overrightarrow{z}$$

But we know that $G\overrightarrow{z} = \overrightarrow{z} = \overrightarrow{y} - A\overrightarrow{p}$:

$$F(A, \overrightarrow{p} + \begin{bmatrix} R \\ I \end{bmatrix} \overrightarrow{z}) = A\overrightarrow{p} + \overrightarrow{y} - A\overrightarrow{p}$$
$$= \overrightarrow{y}$$

But the value $\overrightarrow{x}$ generated by F^{-1} also need to be a small value. To ensure this, the random vector $\overrightarrow{p}$ generated in the first line of F^{-1} must be a small vector and the trapdoor R must be a matrix with small values. When they are small enough, $\overrightarrow{p} + \begin{bmatrix} R \\ I \end{bmatrix} \overrightarrow{z}$ also will be a small vector with high probability. If not, we could try again choosing a different probabilistic $\overrightarrow{p}$.

9.2.1 Choosing G

In an exemple of $n = 2$ and $q = 8$ or some other power of two, we could choose the following G:

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

If we want to discover a value $\overrightarrow{x}$ such that $G\overrightarrow{x} = \overrightarrow{y}$, we could find it choosing $\overrightarrow{x}$ as the binary representation of $\overrightarrow{y}$. For example, if $\overrightarrow{y} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$, then $\overrightarrow{x} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}^T$.

For bigger values of n and q, we just use bigger matrices G built with this method. But instead of using the simpler binary representation above, we can get better results using algorithms of gaussian sampling over lattices.

9.2.2 Choosing the Matrix A and the One-Way Trapdoor Function Security

We want to choose a matrix $A \in \mathbb{Z}_q^{n \times m}$ such that $A \begin{bmatrix} R \\ I \end{bmatrix} = G$.

We can choose then:

$$A = \begin{bmatrix} \overline{A} & G - \overline{A}R \end{bmatrix}$$

when $\overline{A} \in \mathbb{Z}_q^{n \times (m-w)}$ is a random uniform matrix.

Theorem 7 *The Micciancio's one-way trapdoor scheme I is secure if Ajtai's hash function H_A is also an one-way function.*

Proof: For all adversaries $\mathcal{A}$ which try to invert Micciancio's one-way function, we could build a new adversary $\mathcal{B}$ which try to invert Ajtai's hash function.

Adversary $\mathcal{B}$ gets matrix A as a system parameter. Then it sets $\overline{A} = A$, chooses a trapdoor R and sends to adversary $\mathcal{A}$ the public key $\begin{bmatrix}\overline{A} & G - \overline{A}R\end{bmatrix}$.

Adversary $\mathcal{B}$ gets from its challenger a digest $\overrightarrow{t_1} \in \mathbb{Z}_q^n$ such that $\overrightarrow{t_1} = A\overrightarrow{x} \mod q$. It chooses a random $\overrightarrow{m*} \in \mathbb{Z}_q^w$ where w is the number of columns of G. It computes $\overrightarrow{t_2} = (G - \overline{A}R)\overrightarrow{m*}$. Finally, it sends $\overrightarrow{t_1} + \overrightarrow{t_2}$ as the challenge to adversary $\mathcal{A}$ and gets the answer $\overrightarrow{m}$.

Let $\overrightarrow{m} = \overrightarrow{m_1}||\overrightarrow{m_2}$ with $m_1 \in \mathbb{Z}_q^m$. Adversary $\mathcal{B}$ returns $\overrightarrow{m_1}$.

If $\mathcal{A}$ outputs a correct answer, it means that $\overline{A}\overrightarrow{m_1} + (G - \overline{A}R)\overrightarrow{m_2} = \overrightarrow{t_1} + \overrightarrow{t_2}$. Notice that if $m_2 = m*$, then $\overline{A}\overrightarrow{m_1} = \overrightarrow{t_1}$ and so $\overrightarrow{m_1} = \overrightarrow{x}$. So we have:

$$\text{OWadv}[\mathcal{B}, H_B] \geq \text{OWadv}[\mathcal{A}, I] \cdot Pr[\overrightarrow{m_2} = \overrightarrow{m*}]$$

If $\mathcal{A}$ returned a correct answer, it means that m_2 always will be a correct solution to the system of equations represented by $(G - \overline{A}R)\overrightarrow{m_2} = \overrightarrow{t_1} + \overrightarrow{t_2} - \overline{A}\overrightarrow{m_1}$. As $G \in \mathbb{Z}_q^{n \times w}$, the number of solutions is bounded by $w - n + 1$. For the specific G presented in this section, $w = n \log q$.

We can combine these results and get:

$$\text{OWadv}[\mathcal{A}, I] \leq n \log q \cdot \text{OWadv}[\mathcal{B}, A_H]$$

This completes the proof that if Ajtai's hash function H_A is an one-way function, Micciancio's one-way trapdoor function is secure. □

9.3 Combining the Constructions to Create the Chameleon Hash

9.3.1 Security and System Parameters

The security parameter λ here means the number of lines in the matrices for Ajtai's hash function and Micciancio's one-way trapdoor function.

Ajtai's hash matrix will be a fixed random and unifrom $n \times m$ matrix A_1 modulo q which could be the same matrix in all implementations with $m \geq n \log q$. And q is some fixed value chosen to be convenient. It could be some power of two which could represent the numbers which could be stored in 32 or 64 bits. Both A_1 and q are encoded in system paramenter Λ.

9.3.2 Key Generation

The key generator algorithm should generate the one-way trapdoor function's public and private keys. The public key is a matrix $A_2 = \begin{bmatrix}\overline{A} & G - \overline{A}R\end{bmatrix}$

The private key is the trapdoor matrix R. Assuming that we use as G the power-of-two matrix seen in the last section, $G \in \mathbb{Z}_q^{n \times n \log q}$. So R need to have $n \log q$ columns. The number of lines in R depends of the number of columns in the matrix $\overline{A}$. To ensure the same degree of security, we choose the same $m \geq n \log q$ used in the matrix A_1 from Ajtai's hash function. So $R \in \mathbb{Z}_q^{m \times n \log q}$.

Given the trapdoor R, we can compute a random and uniform $\overline{A} \in \mathbb{Z}_q^{n \times m}$. And from this we compute $A_2 \in \mathbb{Z}_q^{n \times mn \log q}$.

The public key is $\mathcal{PK} = A_2$ and the private key is $\mathcal{SK} = (A_2, R)$.

As these matrices could be very large, more realistically we could generate these matrices using some pseudo-random generator G and smaller keys. We could generate $\overline{A}$ from some key k_1 and R from other key k_2. Our private key would be $\mathcal{SK} = (k_1, k_2)$. Unfortunally this compresses just the first m columns of the matrix A_2. The last $n \log q$ columns hide our trapdoor and can't be easily compressed. So our public key would be $\mathcal{PK} = (k_1, G - \overline{A}R)$, meaning that we still need to store a matrix with $n^2 \log q$ elements in the public key.

References

[1] Giuseppe Ateniese and Breno De Medeiros. On the key exposure problem in chameleon hashes. In *International Conference on Security in Communication Networks*, pages 165–179. Springer, 2004.

[2] Mihir Bellare and Todor Ristov. Hash functions from sigma protocols and improvements to vsh. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, pages 125–142, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[3] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. 2020. A draft can be downloaded in https://toc.cryptobook.us/.

[4] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 523–552. Springer, 01 2010.

[5] Hugo Krawczyk and Tal Rabin. Chameleon hashing and signatures, 1997.

[6] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 700–718. Springer, 2012.

[7] David Naccache, David Pointcheval, and Jacques Stern. Twin signatures: an alternative to the hash-and-sign paradigm. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 20–27, 01 2001.

[8] Kaisa Nyberg and Rainer A. Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. In Alfredo De Santis, editor, *Advances in Cryptology — EUROCRYPT'94*, pages 182–193, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.