Hashes Camaleão: Definição e Construção

Thiago Leucz Astrizi

9 de março de 2020

1 Introdução

Hashes camaleão são funções muito semelhantes à hashes comuns. A diferença é que elas possuem uma chave pública e uma chave privada associada a elas. De posse de sua chave pública, é possível usá-las para calcular o hash de algum dado qualquer, o que também permite conferir se o cálculo de um hash está correto. Todas as propriedades características à funções hash criptográficas (resistência à pré-imagem, resistência à segunda pré-imagem, resistência à colisão) estão presentes nas hashes camaleão quando elas são usadas somente junto à sua chave pública.

Tendo acesso à chave privada de uma hash camaleão, a propriedade de resistência à segunda pré-imagem se perde, o que também acaba com a propriedade de resistência à colisões. Dependendo da construção da hash camaleão, a propriedade de resistência à primeira pré-imagem também pdode se perder quando tem-se acesso à chave privada.

2 Hashes Camaleão Simples

Um Esquema de Hash Camaleão Simples CH é uma tupla formada por três algoritmos eficientes $KeyGen$, $Hash$ e $Collision$ tais que:

- **KeyGen** é um algoritmo probabilístico invocado como $(\mathcal{PK}, \mathcal{SK}) \leftarrow^{\$} KeyGen()$ onde $\mathcal{PK}$ é chamado de chave pública e $\mathcal{SK}$ é chamado de chave privada.
- **Hash**: é um algoritmo determinístico invocado como $c \leftarrow Hash(\mathcal{PK}, m, r)$ onde $\mathcal{PK}$ é uma chave pública (como retornada por $KeyGen$), m é uma mensagem, r é um parâmetro aleatório e c é o resumo criptográfico.
- **Collision**: é um algoritmo probabilístico invocado como $r' \leftarrow^{\$} Collision(\mathcal{SK}, m, r, m')$ onde $\mathcal{SK}$ é a chave privada (como retornada por $KeyGen$), m e m' são duas mensagens distintas, r é um parâmetro aleatório e r' retornado é um valor tal que:

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

- Assumimos que as mensagens pertencem à um espaço de mensagens finito M, os parâmetros aleatórios pertencem à um espaço de parâmetros

aleatórios finito R e todos os resumos criptográficos possíveis pertencem ao espaço re resumo criptográfico finito C. Dizemos que $CH = (KeyGen, Hash, Collision)$ é definida sobre (M, R, C).

Podemos avaliar o quão seguros são vários tipos de esquemas de hash camaleão observando as seguintes propriedades deles:

- **Resistência a Colisão:** Para todo adversário eficiente $\mathcal{A}$ a probabilidade do experimento abaixo retornar 1 é negligível:

```
1 Experimento CollRes_A^CH(n):
2 |  (PK, SK) ←$ KeyGen(1^n);
3 |  (m, r, m', r') ← A(PK);
4 |  if Hash(PK, m, r) = Hash(PK, m', r') e (m, r) ≠ (m', r') then
5 |  |  retorne 1;
6 |  else
7 |  |  retorne 0;
8 |  end
9 Fim
```

- **Resistência à Pré-Imagem:** Para todo adversário eficiente $\mathcal{A}$, a probabilidade do experimento abaixo retornar 1 é negligível:

```
1  Experimento PreImg_A^CH(n):
2  |  (PK, SK) ←$ KeyGen(1^n);
3  |  m ←$ M;
4  |  r ←$ R;
5  |  c ← Hash(PK, m, r);
6  |  m' ← A(PK, c, r);
7  |  if m' = m then
8  |  |  retorne 1;
9  |  else
10 |  |  retorne 0;
11 |  end
12 Fim
```

- **Segurança Semântica:** Se para todo adversário eficiente $\mathcal{A}$, a probabilidade do experimento abaixo retornar 1 é negligível:

Experimento $\texttt{SemanticSecurity}_{\mathcal{A}}^{CH}(n)$**:**
  $(\mathcal{PK}, \mathcal{SK}) \leftarrow$**KeyGen**$(1^n)$;
  $(m_0, m_1) \leftarrow \mathcal{A}(\mathcal{PK})$;
  $b \leftarrow^{\$} \{0, 1\}$;
  $r \leftarrow^{\$} R$;
  $c \leftarrow Hash(\mathcal{PK}, m_b)$;
  $b' \leftarrow \mathcal{A}(c, r)$;
  if $b' = b$ **then**
    retorne 1;
  else
    retorne 0;
  end

A segurança semântica implica a resistência à pré-imagem. Se um adversário pode calcular a pré-imagem com probabilidade não-negligível, então ele pode ser usado para construir um adversário que quebra a segurança semântica obtendo a pré-imagem do

- **Livre de Exposição de Chave:** Se para todo adversário eficiente $\mathcal{A}$, capaz de realizar uma quantidade polinomial de consultas à um oráculo que retorna colisões aleatórias no esquema de hash camaleão, a probabilidade do experimento abaixo retornar 1 é negligível.

Experimento $\texttt{KeyExposition}_{\mathcal{A}}^{CH}(n)$**:**
  $(\mathcal{PK}, \mathcal{SK}) \leftarrow$ **KeyGen**(1^n);
  $\mathcal{SK}' \leftarrow \mathcal{A}^{Collision(\mathcal{SK},.,.,.)}(\mathcal{PK})$;
  if $\mathcal{SK} = \mathcal{SK}'$ **then**
    retorne 1;
  else
    retorne 0;
  end
Fim

Se um esquema de hash camaleão é livre de exposição de chaves, a mesma chave pode ser reutilizada mesmo quando colisões obtidas com a sua chave privada tornam-se públicas.

- A **Resistência à Falsificação** afirma que para todo algoritmo adversário $\mathcal{A}$ que recebe uma chave pública $\mathcal{PK}$ e pode consultar um número polinomial de vezes um oráculo que retorna diferentes resultados de $Collision(\mathcal{SK}, m, r, m')$ para quaisquer valores de m, r e m', a probabilidade do experimento abaixo retornar 1 é negligível:

Experimento $\texttt{ForgeRes}_{\mathcal{A}}^{CH}(n)$**:**
  $(\mathcal{PK}, \mathcal{SK}) \leftarrow$ **KeyGen**(1^n);
  $(m, r, m', r') \leftarrow \mathcal{A}^{Collision(\mathcal{SK},.,.,.)}(\mathcal{PK})$;
  if $Hash(\mathcal{PK}, m, r) = Hash(\mathcal{PK}, m', r')$ ***e*** $(m, r) \neq (m', r')$ ***e*** r *e* r' *não foram retornados pelo oráculo* **then**
    retorne 1;
  else
    retorne 0;
  end
Fim

Sem esta propriedade, uma vez que uma colisão é descoberta para uma mensagem m, não é possível garantir a segurança de qualquer outro uso da hash camaleão para calcular novamente o hash desta mesma mensagem. Contudo, a função ainda pode ser utilizada com a mesma chave para calcular o hash de outras mensagens. Na maioria dos casos esta propriedade é desejável, mas existem alguns usos de hash camaleão, como no caso das assinaturas camaleão como propostas por Krawczyk em [4] nos quais a ausência desta propriedade é um requisito.

É importante notar que a Resistência à Falsificação implica que um esquema é Livre de Exposição de Chaves. Afinal, um adversário capaz de computar eficientemente com probabilidade não-negligível a chave privada tendo acesso à um número polinomial de colisões pode ser usado para construir um adversário com uma vantagem não-negligível em encontrar colisões. Também será válida a contrapositiva de que se um esquema não é livre de exposição de chave, ele não será resistente à falsificação.

- Um **Trapdoor de Pré-Imagem** é característica de algumas hashes camaleão onde de posse da chave privada podemos calcular não apenas uma colisão, mas também obter uma pré-imagem. Em outras palavras, existe um algoritmo probabilístico $PreImage(m, c)$ que retorna um valor r tal que $Hash(m, r) = c$.

  Quando tal algoritmo existe, podemos apenas definir ele sem definir explicitamente o algoritmo que calcula colisões por meio da chave privada. Isso porque:

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

É importante notar que a composição de uma hash convencional e uma hash camaleão é uma hash camaleão. Grande parte das definições de hash camaleão assume que a mensagem de entrada possui um tamanho fixo ou está em um formato pré-estabelecido. Em tais casos, assumimos que a mensagem já passou previamente por uma função hash que a deixou no formato adequado.

Uma função de hash camaleão $Hash(\mathcal{PK}, m, r)$ pode ser vista como um caso particular de uma função hash com chave $H_{\mathcal{PK}}(m||r)$ e deste fato podemos derivar algumas propriedades. Se $|M||R| \geq s|T|$ para algum s super-polinomial em relação ao parâmetro de segurança n, então se a função hash é resistente à colisão, então ela será uma função de mão única. Em tais casos, a existência de

uma função $PreImage$ também implica a existência de uma função $Collision$ dada a chave privada.

2.1 Construção Baseada em Permutações de Resíduos Quadráticos (Krawczyk) (2000) [4]

2.1.1 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.

2.1.2 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:

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

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

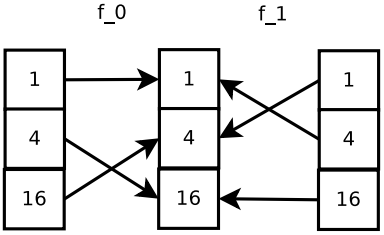
Desta forma, conseguimos fatorar n e ter acesso à chave privada $\mathcal{SK}$.

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

2.1.3 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.

2.1.4 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

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

2.2.1 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 $m + xr = m' + xr'$.

2.2.2 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.

2.2.3 Medida de Desempenho

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

2.3 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.

2.3.1 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:

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

2.3.2 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 [6] 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 $Hash(m, PreImage(c, m)) = c$.

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

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 $Hash(m, PreImage(c, m)) = c$.

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

2.3.3 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 [5], no Apêndice A.

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

2.3.4 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

2.4 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.

2.4.1 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 $Hash(m, PreImage(c, m)) = c$.

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

2.4.2 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 [5].

A segurança semântica deste esquema ainda está para ser demonstrada.

2.4.3 Medida de Desempenho

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

2.5 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.

2.5.1 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:

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

2.5.2 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.

2.5.3 Medida de Desempenho

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

2.6 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.

2.6.1 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.

2.7 Construção Baseada em Reticulados (David Cash) (2010) [3]

A segurança deste esquema não se baseia em fatoração de números ou no logaritmo discreto, mas no Problema da Solução Inteira Pequena (SIS), um problema baseado em reticulados. Isso torna este esquema seguro mesmo diante de computadores quânticos até onde se sabe.

A segurança desta construção se baseia no fato de que é difícil encontrar qualquer vetor x inteiro cuja norma euclideana é suficientemente pequena tal que $Ax = b$, onde A é uma matriz inteira retangular suficientemente grande em termos de número de colunas e b é um vetor conhecido.

A construção utiliza sempre matrizes e vetores inteiros módulo q nas definições seguintes.

2.7.1 Funções

Keygen: Seja $H \in \mathbb{Z}^{n \times n}$ uma matriz inversível qualquer, podendo ser uma matriz identidade e G uma matriz selecionada previamente tal que para G é conhecido método eficiente de obter valores de x com norma euclideana pequena tais que $Gx = b$.

Como chave privada, gere uma matriz aleatória $R \in \mathbb{Z}^{w \times n}$. Os valores dela devem ser pequenos para aumentar a qualidade do *trapdoor*.

Como chave pública gere a matriz $A = [\overline{A}|HG - \overline{A}R]$, onde $\overline{A}$ é uma submatriz inteira gerada aleatoriamente.

O objetivo desta construção é que com isso geramos matrizes tais que:

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

Hash: Seja $\mathcal{H}$ uma função hash convencional tal que a saída dela possa ser interpretada como um vetor inteiro módulo q com mesmo número de elementos que o número de linhas de A. Definimos a função hash do esquema como:

$$Hash(\mathcal{PK} = A, m, r) = \mathcal{H}(m) + Ar \mod q$$

Collision: Como queremos encontrar uma colisão, dado m, r e m', queremos encontrar um valor r' de norma euclideana pequena tal que:

$$\mathcal{H}(m) + Ar \equiv \mathcal{H}(m') + Ar' \pmod q$$

O valor de r' é então uma solução para a equação $Ax = \mathcal{H}(m) - \mathcal{H}(m') + Ar \mod q$.

Par obter uma solução para esta equação utiliza-se um vetor qualquer de norma euclideana pequena p e obtém-se o valor x' tal que ele seja a solução para a equação abaixo:

$$Gx' = H^{-1}(\mathcal{H}(m) - \mathcal{H}(m') + Ar - Ap) \mod q$$

Dados os valores de x' e p, a nossa função $Collision$ é definida como:

$$Collision(\mathcal{SK} = R, m, r, m') = p + \begin{bmatrix} R \\ I \end{bmatrix} x' \mod q$$

Prova:

Para provar que o resultado obtido é correto, basta mostrarmos que o valor retornado pela função acima é uma solução para $Ax = \mathcal{H}(m) - \mathcal{H}(m') + Ar \mod q$. Se multiplicarmos o resultado da função de colisão pela matriz A obtemos:

$$A(p + \begin{bmatrix} R \\ I \end{bmatrix} x') = Ap + A \begin{bmatrix} R \\ I \end{bmatrix} x'$$

Temos que $A \begin{bmatrix} R \\ I \end{bmatrix} = HG$:

$$Ap + A \begin{bmatrix} R \\ I \end{bmatrix} x' = Ap + HGx'$$

Sabemos também que $Gx' = H^{-1}(\mathcal{H}(m) - \mathcal{H}(m') + Ar - Ap)$, e portanto:

$$Ap + HGx' = Ap + HH^{-1}(\mathcal{H}(m) - \mathcal{H}(m') + Ar - Ap)$$

Após multiplicar H pela sua inversa e dos dis termos Ap se anularem, obtemos o resultado que queríamos para mostrar a corretude da função de colisão.

2.7.2 Detalhes da Construção

Para garantir a segurança do esquema, o número de colunas da matriz A de n linhas deve ser ao menos $n \log q$. Além disso, só devemos considerar um vetor como tendo uma norma euclideana suficientemente pequena para ser aceito como solução quando a norma dele é no máximo um valor próximo de $\sqrt{n \log q}$.

Construção da matriz G:

Seja $q = 2^k$. Seja o vetor $g = [1, 2, 4, 8, \ldots, 2^{k-1}]$. Podemos construir a matriz G então como:

$$\begin{bmatrix} g & \cdots & \cdots & \cdots \\ \cdots & g & \cdots & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots & g \end{bmatrix}$$

Desta forma, podemos encontrar uma solução para uma equação $Gx = u \mod q$ simplesmente escolhendo x como a representação binária de u. Esta é uma forma viável de gerar vetores suficientemente pequenos apenas quando temos uma construção com um valor muito pequeno de n.

Para os demias casos, o modo recomendado de obter uma solução é gerar um reticulado com uma base formada pela matriz esparsa preenchida por 2 na diagonal e com um valor -1 abaixo de cada diagonal e obter um vetor suficientemente pequeno por meio de algoritmos de amostragem gaussiana.

2.7.3 Medida de Desempenho

KeyGen	–	Hash	0,0017129s	Collision	–

Também é importante mencionar que para valores consifderados seguros, o tamanho da chave privada nesta construção chega a 613KB. A chave pública é seis vezes menor e pode ser reduzida muito mais por meio das mesmas técnicas utilizadas para reduzir tamanhos de chave de funções hash baseados em reticulados.

2.8 Características dos Esquemas Apresentados

A tabela abaixo sintetiza as características mais relevantes dos esquemas vistos nesta seção. Cada esquema está identificado pelo número da subseção em que ele foi apresentado.

Esquema	2.1	2.2	2.3	2.4	2.5	2.6	2.7
Livre de Exposição de Chave	X	X	✓	✓	X	X	✓
Resistência à Falsificação	X	X	✓	✓	X	X	✓
Trapdoor de Pré-Imagem	✓	X	✓	✓	✓	X	X

Referências

[1] Giuseppe Ateniese and Breno de Medeiros. On the key exposure problem in chameleon hashes. In Carlo Blundo and Stelvio Cimato, editors, *Security in Communication Networks*, pages 165–179, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[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] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. volume 2010, pages 523–552, 01 2010.

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

[5] David Naccache, David Pointcheval, and Jacques Stern. Twin signatures: an alternative to the hash-and-sign paradigm. pages 20–27, 01 2001.

[6] 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.