An introduction to Hash functions

Anna Rimoldi

eRISCS - Universitée de la Méditerranée, Marseille

Secondo Workshop di Crittografia BunnyTN 2011

An overview

- Hash functions are means securely reduce a string of arbitrarily length into a fixed length digit.

An overview

- Hash functions are means securely reduce a string of arbitrarily length into a fixed length digit.
- The main problem is the definition of securely.

An overview

- Hash functions are means securely reduce a string of arbitrarily length into a fixed length digit.
- The main problem is the definition of securely.
- Use of hash function: signature scheme, store password files, key derivation function, tags of files to detect changes, inside PRNGs, inside protocols, etc...

Classical definitions

Let $\mathcal{X}$ be the set of all possible messages. Let $\mathcal{Y}$ be the set of all possible message digests (or authentication tags). Let $\mathcal{K}$ be the set of all possible keys.

(keyed) Hash function

For any key k in the key-space $\mathcal{K}$, we define (keyed) hash function as the function

$$h_k : \mathcal{X} \to \mathcal{Y}.$$

(unkeyed) Hash function

An unkeyed hash function is a function $h_k : \mathcal{X} \to \mathcal{Y}$, where $k \in \mathcal{K}$ but $|\mathcal{K}| = 1$, i.e. there is only a possible key.

Classical definitions

Let $\mathcal{X}$ be the set of all possible messages. Let $\mathcal{Y}$ be the set of all possible message digests (or authentication tags). Let $\mathcal{K}$ be the set of all possible keys.

(keyed) Hash function

For any key k in the key-space $\mathcal{K}$, we define (keyed) hash function as the function

$$h_k : \mathcal{X} \rightarrow \mathcal{Y}.$$

(unkeyed) Hash function

An unkeyed hash function is a function $h_k : \mathcal{X} \rightarrow \mathcal{Y}$, where $k \in \mathcal{K}$ but $|\mathcal{K}| = 1$, i.e. there is only a possible key.

The set $\mathcal{X}$ could be a finite or an infinite set.
We will always assume that $|\mathcal{X}| \geq |\mathcal{Y}|$.
In practical situation, we will assume the stronger condition $|\mathcal{X}| \geq 2|\mathcal{Y}|$.
Moreover, a common choice for $|\mathcal{Y}|$ consist of having at least 160-bit of message digests.

Valid pair

A pair $(\bar{x}, \bar{y}) \in \mathcal{X} \times \mathcal{Y}$ is said to be a valid pair under the key k if $h_k(\bar{x}) = \bar{y}$.

Obviously, it is convenient to prevent the construction of certain types of valid pairs by an adversary.

Classical security requirement

If an hash function is to be considered secure, it should be the case that these three problems are difficult to solve:

1. **Preimage:** given $h : \mathcal{X} \to \mathcal{Y}$ and $\bar{y} \in \mathcal{Y}$, is difficult to find $\bar{x} \in \mathcal{X}$ such that $h(\bar{x}) = \bar{y}$.

In case 1 we say that the hash function is one-way or Preimage resistant;

Classical security requirement

If an hash function is to be considered secure, it should be the case that these three problems are difficult to solve:

1. **Preimage:** given $h : \mathcal{X} \to \mathcal{Y}$ and $\bar{y} \in \mathcal{Y}$, is difficult to find $\bar{x} \in \mathcal{X}$ such that $h(\bar{x}) = \bar{y}$.
2. **Second Preimage:** given $h : \mathcal{X} \to \mathcal{Y}$ and $\bar{x} \in \mathcal{X}$, is difficult to find $x^* \in \mathcal{X}$ (with $x^* \neq \bar{x}$) such that $h(x^*) = h(\bar{x})$.

In case 1 we say that the hash function is one-way or Preimage resistant; in case 2 we say that h is Second Preimage resistant;

Classical security requirement

If an hash function is to be considered secure, it should be the case that these three problems are difficult to solve:

1. **Preimage:** given $h : \mathcal{X} \rightarrow \mathcal{Y}$ and $\bar{y} \in \mathcal{Y}$, is difficult to find $\bar{x} \in \mathcal{X}$ such that $h(\bar{x}) = \bar{y}$.
2. **Second Preimage:** given $h : \mathcal{X} \rightarrow \mathcal{Y}$ and $\bar{x} \in \mathcal{X}$, is difficult to find $x^* \in \mathcal{X}$ (with $x^* \neq \bar{x}$) such that $h(x^*) = h(\bar{x})$.
3. **Collision:** given $h : \mathcal{X} \rightarrow \mathcal{Y}$, is difficult to find $x^*, \bar{x} \in \mathcal{X}$ (with $x^* \neq \bar{x}$) such that $h(x^*) = h(\bar{x})$.

In case 1 we say that the hash function is one-way or Preimage resistant;
in case 2 we say that h is Second Preimage resistant;
in case 3 we say that h is Collision resistant.

Security hypothesis for specific applications

As hash functions are widely used, various requirements are needed to ensure the security of construction based on hash functions.

- Collision resistance $\rightarrow$ signatures, MACs.
- Second Preimage resistance $\rightarrow$ signatures.
- Preimage resistance $\rightarrow$ signatures , password files, bit commitment (for hiding).
- Pseudo Random Functions $\rightarrow$ key derivation, MACs.
- Pseudo Random Oracle $\rightarrow$ protocols, PRNGs.

We want the hash function to behave in a way which would prevent any attacker from doing anything malicious to inputs to the hash function:

- One-wayness (no inversion).
- No collisions (up to the birthday bound).
- No second preimages.
- Outputs which are “well” distributed.

We want the hash function to behave in a way which would prevent any attacker from doing anything malicious to inputs to the hash function:

- One-wayness (no inversion).
- No collisions (up to the birthday bound).
- No second preimages.
- Outputs which are “well” distributed.

Therefore the ideal hash function attaches for each possible message x a random value as $h(x)$.

We want the hash function to behave in a way which would prevent any attacker from doing anything malicious to inputs to the hash function:

- One-wayness (no inversion).
- No collisions (up to the birthday bound).
- No second preimages.
- Outputs which are "well" distributed.

Therefore the ideal hash function attaches for each possible message x a random value as $h(x)$.

If an hash function is well designed, it should be the case that the only efficient way to determined the value $h(x)$ for a given x is to actually evaluate the function h at the value x. This should remain true even if many other values $h(x_1), h(x_2), \ldots,$ have already been computed.

Ideal Model

Bellare and Rogaway introduced a mathematical model of an IDEAL hash function:

Random Oracle Model

$h : \mathcal{X} \to \mathcal{Y}$ is chosen randomly from the set of all possible hash functions. We are only permitted oracle access to the function h.

Ideal Model

Bellare and Rogaway introduced a mathematical model of an IDEAL hash function:

Random Oracle Model

$h : \mathcal{X} \to \mathcal{Y}$ is chosen randomly from the set of all possible hash functions.
We are only permitted oracle access to the function h.

This means that we are not given a formula or an algorithm to compute values of h. The only way to compute $h(x)$ is to query the oracle.

Ideal Model

Bellare and Rogaway introduced a mathematical model of an IDEAL hash function:

Random Oracle Model

$h : \mathcal{X} \to \mathcal{Y}$ is chosen randomly from the set of all possible hash functions. We are only permitted oracle access to the function h.

This means that we are not given a formula or an algorithm to compute values of h. The only way to compute $h(x)$ is to query the oracle.

Although a true random oracle does not exists in real life, we hope that a well designed hash function will behave like a random oracle.

Ideal Model

Bellare and Rogaway introduced a mathematical model of an IDEAL hash function:

Random Oracle Model

$h : \mathcal{X} \rightarrow \mathcal{Y}$ is chosen randomly from the set of all possible hash functions. We are only permitted oracle access to the function h.

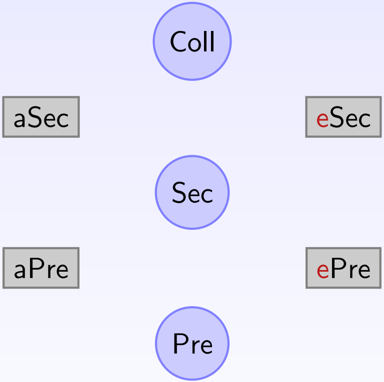
This means that we are not given a formula or an algorithm to compute values of h. The only way to compute $h(x)$ is to query the oracle.

Although a true random oracle does not exists in real life, we hope that a well designed hash function will behave like a random oracle.

It is useful to study the random oracle model and its security w.r.t. the three problems introduced above.

Properties Relations

Properties Relations



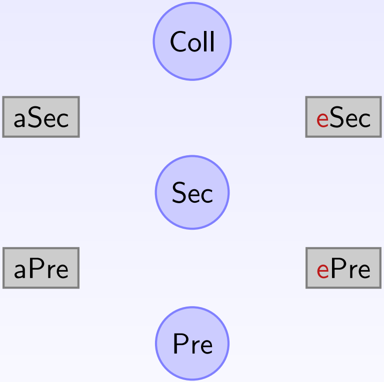
"Everywhere" notion

In this contest, the adversary selects the challenge and it is then a randomly chosen key.

"Always" notion

In this contest, the adversary selects the key and it is then a randomly chosen challenge.

Properties Relations



"Everywhere" notion

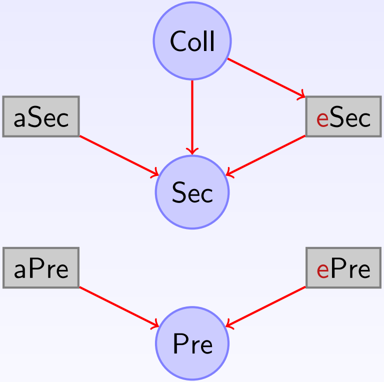
In this contest, the adversary selects the challenge and it is then a randomly chosen key.

"Always" notion

In this contest, the adversary selects the key and it is then a randomly chosen challenge.

Rogaway and Shrimpton (2004) considered the implications (or lack of implication) between all seven security notions.

Properties Relations



"Everywhere" notion

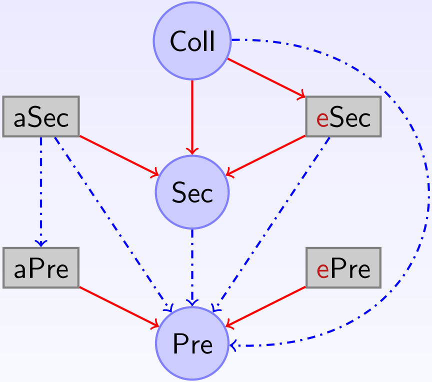
In this contest, the adversary selects the challenge and it is then a randomly chosen key.

"Always" notion

In this contest, the adversary selects the key and it is then a randomly chosen challenge.

Rogaway and Shrimpton (2004) considered the implications (or lack of implication) between all seven security notions.

Properties Relations



"Everywhere" notion

In this contest, the adversary selects the challenge and it is then a randomly chosen key.

"Always" notion

In this contest, the adversary selects the key and it is then a randomly chosen challenge.

Rogaway and Shrimpton (2004) considered the implications (or lack of implication) between all seven security notions.

CONSTRUCTION OF HASH FUNCTIONS

Iterated Hash function

It was understood that a hash function h should be constructed by iterating a compression function f with fixed size inputs.

Let m, ℓ and t be positive integers, with $t \geq 1$.
Let $f : (\mathbb{F}_2)^{m+t} \to (\mathbb{F}_2)^m$ be a compression function. We can construct an iterated hash function $h : \mathcal{X} \to \mathcal{Y}$ based on f where

$$\mathcal{X} = \bigcup_{i=m+t+1}^{\infty} (\mathbb{F}_2)^i \quad \text{and} \quad \mathcal{Y} = (\mathbb{F}_2)^\ell.$$

Iterated Hash function

It was understood that a hash function h should be constructed by iterating a compression function f with fixed size inputs.

Let m, ℓ and t be positive integers, with $t \geq 1$.
Let $f : (\mathbb{F}_2)^{m+t} \to (\mathbb{F}_2)^m$ be a compression function. We can construct an iterated hash function $h : \mathcal{X} \to \mathcal{Y}$ based on f where

$$\mathcal{X} = \bigcup_{i=m+t+1}^{\infty} (\mathbb{F}_2)^i \quad \text{and} \quad \mathcal{Y} = (\mathbb{F}_2)^{\ell}.$$

$$\begin{aligned} H_0 &= IV \\ H_i &= f(y_i, H_{i-1}) \\ h(x) &= g(H_t) \end{aligned}$$

The evaluation of h consists of the following three main steps.

- **Preprocessing:** given $\bar{x} \in \mathcal{X}$ s.t. $|\bar{x}| \geq m + t + 1$, using a public algorithm, we construct an element $y \in (\mathbb{F}_2)^{rt}$ (in fact we require that $|y| = rt \geq |x|$ because the injectivity) as follows $y = y_1||y_2|| \cdots ||y_r$, where $|y_i| = t$ for $1 \leq i \leq r$.

The evaluation of h consists of the following three main steps.

- **Preprocessing:** given $\bar{x} \in \mathcal{X}$ s.t. $|\bar{x}| \geq m+t+1$, using a public algorithm, we construct an element $y \in (\mathbb{F}_2)^{rt}$ (in fact we require that $|y| = rt \geq |x|$ because the injectivity) as follows $y = y_1||y_2||\cdots||y_r$, where $|y_i| = t$ for $1 \leq i \leq r$.
- **Processing:** given a public initial value $IV \in (\mathbb{F}_2)^m$, we construct a sequence of elements in $(\mathbb{F}_2)^m$, that we call $z_0, \ldots, z_r$ as follows:

$$
\begin{aligned}
z_0 &:= IV \\
z_1 &:= f(z_0||y_1) \\
z_2 &:= f(z_1||y_2) \\
&\vdots \\
z_r &:= f(z_{r-1}||y_r).
\end{aligned}
$$

The evaluation of h consists of the following three main steps.

- **Preprocessing:** given $\bar{x} \in \mathcal{X}$ s.t. $|\bar{x}| \geq m + t + 1$, using a public algorithm, we construct an element $y \in (\mathbb{F}_2)^{rt}$ (in fact we require that $|y| = rt \geq |x|$ because the injectivity) as follows $y = y_1||y_2|| \cdots ||y_r$, where $|y_i| = t$ for $1 \leq i \leq r$.
- **Processing:** given a public initial value $IV \in (\mathbb{F}_2)^m$, we construct a sequence of elements in $(\mathbb{F}_2)^m$, that we call $z_0, \ldots, z_r$ as follows:

$$
\begin{aligned}
z_0 &:= IV \\
z_1 &:= f(z_0||y_1) \\
z_2 &:= f(z_1||y_2) \\
&\vdots \\
z_r &:= f(z_{r-1}||y_r).
\end{aligned}
$$

- **Output transformation:** using a public function $g = (\mathbb{F}_2)^m \to (\mathbb{F}_2)^\ell$ we compute $g(z_r)$ obtaining $h(\bar{x}) \in (\mathbb{F}_2)^\ell$.

Comments

- A commonly used preprocessing step consist of constructing $y \in (\mathbb{F}_2)^{rt}$ using a *padding function* $\mathbf{pad}(x)$: $y = x||\mathbf{pad}(x)$.
- $\mathbf{pad}(x)$ typically incorporates the value of $|x|$ and pads the result with additional so that the resulting string y has length exactly rt.

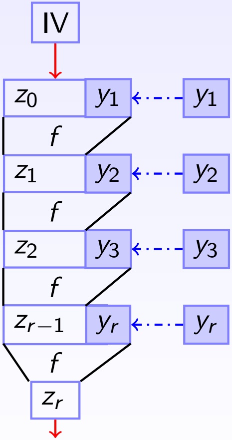
Comments

- A commonly used preprocessing step consist of constructing $y \in (\mathbb{F}_2)^{rt}$ using a *padding function* $\mathbf{pad}(x)$: $y = x||\mathbf{pad}(x)$.
- $\mathbf{pad}(x)$ typically incorporates the value of $|x|$ and pads the result with additional so that the resulting string y has length exactly rt.
- The preprocessing step must ensure that the mapping $x \mapsto y$ is an injection. If it is not one-to-one, then it may be possible to find $x \neq x'$ so that $y = y'$. Then $h(x) = h(x')$ is not collision resistant.

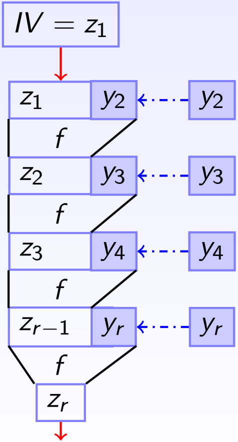
Comments

- A commonly used preprocessing step consist of constructing $y \in (\mathbb{F}_2)^{rt}$ using a *padding function* $\mathbf{pad}(x)$: $y = x||\mathbf{pad}(x)$.
- $\mathbf{pad}(x)$ typically incorporates the value of $|x|$ and pads the result with additional so that the resulting string y has length exactly rt.
- The preprocessing step must ensure that the mapping $x \mapsto y$ is an injection. If it is not one-to-one, then it may be possible to find $x \neq x'$ so that $y = y'$. Then $h(x) = h(x')$ is not collision resistant.
- It is also easy to see that the absence of an output transformation leads to an extension attack, that is, one can compute $h(x||y)$ from $h(x)$ and y, without knowing x, which is undesirable for some applications.

Comments (2)



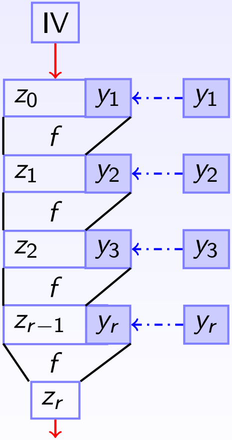
IV
z_0 y_1 y_1
f
z_1 y_2 y_2
f
z_2 y_3 y_3
f
z_{r-1} y_r y_r
f
z_r

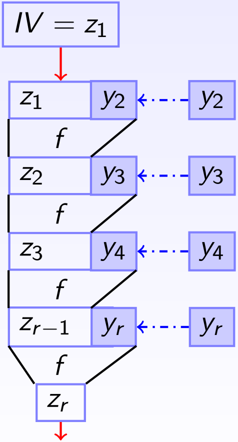


$IV = z_1$
z_1 y_2 y_2
f
z_2 y_3 y_3
f
z_3 y_4 y_4
f
z_{r-1} y_r y_r
f
z_r

Comments (2)

Iterating f can degraded its security: a trivial example is Second Preimage





Merkle-Damgard construction

The Merkle-Damgard construction is an iterated hash function which permits a formal security proof to be given.

Theorem

Let $f : (\mathbb{F}_2)^{m+t} \to (\mathbb{F}_2)^m$ be a collision resistant compression function, where $t \geq 1$. Then there exists a collision resistant hash function $h : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} = \bigcup_{i=m+t+1}^{\infty} (\mathbb{F}_2)^i$ and $\mathcal{Y} = (\mathbb{F}_2)^m$.

Merkle-Damgard construction

The Merkle-Damgard construction is an iterated hash function which permits a formal security proof to be given.

Theorem

Let $f : (\mathbb{F}_2)^{m+t} \to (\mathbb{F}_2)^m$ be a collision resistant compression function, where $t \geq 1$. Then there exists a collision resistant hash function $h : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} = \bigcup_{i=m+t+1}^{\infty} (\mathbb{F}_2)^i$ and $\mathcal{Y} = (\mathbb{F}_2)^m$.

Moreover the number of times f is computed in the evaluation of h is at most

$$\begin{array}{rll} 1 + \left\lceil \dfrac{n}{t-1} \right\rceil & \text{if} & t \geq 2 \\ 2n+2 & \text{if} & t = 1 \end{array}$$

where $|x| = n$.

In other words, (in case $t \geq 2$)

given our collision resistant compression function f,
if the padding contains the length of the input string
and if f is Preimage resistant,

$$\Downarrow$$

the iterated hash function based on f will be a collision resistant hash function.

Improving MD iteration

- Multi collision attack and impact on concatenation (Joux 2004)
- Long message Second Preimage arrack (Kelsey and Schneier 2005)
- Herding attack (Kelsey Kohono 2006)
- salt + output transformation + counter + wide pipe

Ideals vs standard

Standard model proofs

Consider standard (real world) functionalities
Often results in inefficient (or not provable) scheme

Ideals vs standard

Standard model proofs

Consider standard (real world) functionalities

Often results in inefficient (or not provable) scheme

Ideal Model Proof

Better than ad hoc design

More efficient schemes

Excludes "generics" attack

Uses ideal functionalities: random oracles, ideal block ciphers/permutations

Weaker security guarantee than standard model.

Unfortunately, very few hash functions are designed based on a strong compression function.

COMPRESSION FUNCTION

- based on block ciphers
- permutations
- based on arithmetic primitive

(Iterated) block ciphers

Let $V = \mathcal{P} = \mathcal{C} = (\mathbb{F}_q)^n$ be the space of all possible messages.
Let $\mathcal{K} = (\mathbb{F}_q)^\ell$ be the space of all possible keys.

(Iterated) block ciphers

Let $V = \mathcal{P} = \mathcal{C} = (\mathbb{F}_q)^n$ be the space of all possible messages.
Let $\mathcal{K} = (\mathbb{F}_q)^\ell$ be the space of all possible keys.

Definition

We say that $\phi : \mathcal{P} \times \mathcal{K} \to \mathcal{C}$ is an **(algebraic) block cipher** if, for any $k \in \mathcal{K}$, the function

$$\phi_k : V \to V, \quad \phi_k(x) = \phi(x, k).$$

is a permutation of V.

(Iterated) block ciphers

Let $V = \mathcal{P} = \mathcal{C} = (\mathbb{F}_q)^n$ be the space of all possible messages.
Let $\mathcal{K} = (\mathbb{F}_q)^\ell$ be the space of all possible keys.

Definition

We say that $\phi : \mathcal{P} \times \mathcal{K} \to \mathcal{C}$ is an **(algebraic) block cipher** if, for any $k \in \mathcal{K}$, the function

$$\phi_k : V \to V, \quad \phi_k(x) = \phi(x, k).$$

is a permutation of V.

Any key $k \in \mathcal{K}$ specifies an encryption function ϕ_k.

(Iterated) block ciphers

Let $V = \mathcal{P} = \mathcal{C} = (\mathbb{F}_q)^n$ be the space of all possible messages.
Let $\mathcal{K} = (\mathbb{F}_q)^\ell$ be the space of all possible keys.

Definition

We say that $\phi : \mathcal{P} \times \mathcal{K} \to \mathcal{C}$ is an **(algebraic) block cipher** if, for any $k \in \mathcal{K}$, the function

$$\phi_k : V \to V, \quad \phi_k(x) = \phi(x, k).$$

is a permutation of V.

Any key $k \in \mathcal{K}$ specifies an encryption function ϕ_k.

Iterated: the encryption proceeds through N rounds: $\phi_k = \tau_k^1 \circ \cdots \circ \tau_k^N$

(Iterated) block ciphers

Let $V = \mathcal{P} = \mathcal{C} = (\mathbb{F}_q)^n$ be the space of all possible messages.
Let $\mathcal{K} = (\mathbb{F}_q)^\ell$ be the space of all possible keys.

Definition

We say that $\phi : \mathcal{P} \times \mathcal{K} \to \mathcal{C}$ is an **(algebraic) block cipher** if, for any $k \in \mathcal{K}$, the function

$$\phi_k : V \to V, \quad \phi_k(x) = \phi(x, k).$$

is a permutation of V.

Any key $k \in \mathcal{K}$ specifies an encryption function ϕ_k.

Iterated: the encryption proceeds through N rounds: $\phi_k = \tau_k^1 \circ \cdots \circ \tau_k^N$

Any round function τ_k^h: $\gamma\lambda\sigma_k$ where γ is a non linear function

Block cipher based

The first construction for hash functions were all based on block ciphers (in particular based on DES).

Advantages:

confidence of he community in a block cipher design

very compact implementation.

small deviation from being ideal can result in devastating attacks on Hash functions based block ciphers

Block cipher based

The first construction for hash functions were all based on block ciphers (in particular based on DES).

Advantages:

confidence of he community in a block cipher design

very compact implementation.

Problems:

how to construct hash function with a result larger than the block length (not sufficient to obtain collision resistance)

small deviation from being ideal can result in devastating attacks on Hash functions based block ciphers

Block cipher based

The first construction for hash functions were all based on block ciphers (in particular based on DES).

Advantages:

confidence of he community in a block cipher design

very compact implementation.

Problems:

how to construct hash function with a result larger than the block length (not sufficient to obtain collision resistance)

small deviation from being ideal can result in devastating attacks on Hash functions based block ciphers

12 secure construction $\rightarrow$ security proof in ideal model:

Matyas Meyer Oseas: $H_i = \phi_{H_{i-1}}(x) \oplus x_i$

Miyaguchi-Preneel: $H_i = \phi_{H_{i-1}}(x) \oplus x_i \oplus H_{i-1}$

Davies Meyer: $H_i = \phi_{x_i}(H_{i-1})(x) \oplus H_{i-1}$

- SPONGE: Panama, RadioGatun, Keccak,...
- SMALL PERMUTATION:JH, Groestl

If the permutation π is an ideal function, then Sponge is indifferentiable from a Random Oracle.

Other primitive Hash function

Advantages:

sometimes is possible to prove security reductions

compact implementation

Disadvantages:

mathematical structure can be exploited

sometimes slow (exponentiation)

vulnerable to trapdoors

SHA 3

Candidate

	Block cipher	Permutation
Blake	x	
Groestl		2-permutation
JH		x
Keccak		Sponge
Skein	x	
BMW	x	
Cubehash		Sponge
ECHO		x
Fugue		Sponge
Hamsi		x
Luffa		Sponge
Shabal		Sponge
Shavite-3	Davies-Mayer	
SIMD	x	

Thank you for your attention!