Sound Hashing Modes of Arbitrary Functions, Permutations, and Block Ciphers (SoK)

Joan Daemen\textsuperscript{1}  Bart Mennink\textsuperscript{1}  Gilles Van Assche\textsuperscript{2}

Fast Software Encryption
Paris, March 2019

\textsuperscript{1}Radboud University
\textsuperscript{2}STMicroelectronics
Hash function example 1: SHA-256
Hash function example 1: SHA-256

Hash function $h$ from compression function $F$ with Merkle-Damgård:

$$
\begin{align*}
    & M_1 \rightarrow F \\
    & M_2 \rightarrow F \\
    & M_3 \rightarrow F \\
    & M_4 \rightarrow F \\
    & IV \rightarrow F \\
    & digest
\end{align*}
$$
Hash function example 1: SHA-256

Hash function $h$ from compression function $F$ with Merkle-Damgård:

Compression function $F$ from block cipher $B$ with Davies-Meyer:
Hash function example 1: SHA-256

Hash function $h$ from compression function $F$ with Merkle-Damgård:

![Diagram of hash function]

Compression function $F$ from block cipher $B$ with Davies-Meyer:

![Diagram of compression function]

Underlying primitive: block cipher with 256-bit block and 512-bit key
Example 2: MD6 [Rivest et al. 2008]

Hash function $h$ from CF with dedicated tree hash mode:

- 1024-bit intermediate (chaining) values;
- Root output chopped to desired length
- Location (level,index) input to each node

<table>
<thead>
<tr>
<th>level</th>
<th>(2,2)</th>
<th>(2,0)</th>
<th>(2,1)</th>
<th>(2,3)</th>
</tr>
</thead>
</table>

CF from permutation $P$ with dedicated construction:

- Prepend Constant + Map + Chop
- $N(N)$
- $C\pi$ 1-1 map $\pi$
- $\text{const}$ key+UV data
- 15 8+2 64 89 words 89 words 16 words

Prepend Map
Chop

Underlying primitive: 5696-bit permutation
Hash function $h$ from CF $F$ with dedicated tree hash mode:

- 1024-bit intermediate (chaining) values
- Root output chopped to desired length
- Location (level,index) input to each node

```
0
1
2
3
level
```

- Location (level,index) input to each node

```
(2,2) (2,0) (2,1) (2,3)
```

Underlying primitive: 5696-bit permutation
Hash function $h$ from CF $F$ with dedicated tree hash mode:

- 1024-bit intermediate (chaining) values;
- Root output chopped to desired length;
- Location (level,index) input to each node;

CF $F$ from permutation $P$ with dedicated construction:

1. Prepend Constant
2. Map
3. Chop

Underlying primitive: 5696-bit permutation

Example 2: MD6 [Rivest et al. 2008]
Example 2: MD6 [Rivest et al. 2008]

Hash function $h$ from CF $F$ with dedicated tree hash mode:

Underlying primitive: 5696-bit permutation
Example 3: KangarooTwelve [Keccak Team 2016]
Parallel XOF from XOF with Sakura-encoded [KT 2014] tree hash mode:
Example 3: KangarooTwelve [Keccak Team 2016]

Parallel XOF from XOF with Sakura-encoded [KT 2014] tree hash mode:

```
S0 110* CV CV CV … CV CV n-1 FFFF 01
```

XOF from permutation with 

```
M pad trunc Z
```

[KT 2008]:
Example 3: KangarooTwelve [Keccak Team 2016]

Parallel XOF from XOF with Sakura-encoded [KT 2014] tree hash mode:

XOF from permutation with sponge [KT 2008]:

Underlying primitive: 1600-bit permutation KECCAK-p[12]
We cannot prove a hash function $h$ is secure.
We cannot prove a hash function $h$ is secure

Trust in security based on public scrutiny and cryptanalysis
We cannot prove a hash function $h$ is secure

Trust in security based on public scrutiny and cryptanalysis

But we can prove security of idealized version $\mathcal{H}$ of the function

- ... $\mathcal{H}$ is $h$ with underlying primitive replaced by random one
We cannot prove a hash function \( h \) is secure

Trust in security based on public scrutiny and cryptanalysis

But we can prove security of idealized version \( \mathcal{H} \) of the function

- ... \( \mathcal{H} \) is \( h \) with underlying primitive replaced by random one

Ideal hash function: random oracle \( \text{RO} \)

Upper bound on advantage of distinguishing \( \mathcal{H} \) from \( \text{RO} \)

- this bound says something about the mode only
- better attacks must exploit specific properties of primitive
We cannot prove a hash function $h$ is secure

Trust in security based on public scrutiny and cryptanalysis

But we can prove security of idealized version $\mathcal{H}$ of the function

- ... $\mathcal{H}$ is $h$ with underlying primitive replaced by random one

Ideal hash function: random oracle $\mathcal{RO}$

Upper bound on advantage of distinguishing $\mathcal{H}$ from $\mathcal{RO}$

- this bound says something about the mode only
- better attacks must exploit specific properties of primitive

In other words, they bound the success probability of generic attacks
What can happen if you don’t have a good bound?

Length extension property of MAC function

\( h(K, M) \)

not secure against forgery

Fixing requires adding expensive construction: HMAC

Attacks with less complexity than expected

2nd pre-image for long messages

multi-collisions

herding attack, …

Affect all old-style hash standards: MD5, SHA-1 and all SHA-2
What can happen if you don’t have a good bound?

IV

$M_1$

$F$

$M_2$

$cv$

$F$

$M_3$

$cv$

$F$

$M_4$

$pad$

$cv$

$F$

digest
What can happen if you don’t have a good bound?

- Length extension property

<table>
<thead>
<tr>
<th>M1</th>
<th>F</th>
<th>M2</th>
<th>F</th>
<th>M3</th>
<th>F</th>
<th>M4</th>
<th>pad</th>
<th>F</th>
<th>digest</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV</td>
<td>F</td>
<td>cv</td>
<td>F</td>
<td>cv</td>
<td>F</td>
<td>cv</td>
<td>F</td>
<td>cv</td>
<td>F</td>
</tr>
</tbody>
</table>
What can happen if you don’t have a good bound?

- Length extension property
  - MAC function $h(K|M)$ not secure against forgery
What can happen if you don’t have a good bound?

- **Length extension property**
  - MAC function $h(K|M)$ not secure against forgery
  - fixing requires adding expensive construction: HMAC
What can happen if you don’t have a good bound?

- Length extension property
  - MAC function $h(K|M)$ not secure against forgery
  - fixing requires adding expensive construction: HMAC

- Attacks with less complexity than expected
  - 2nd pre-image for long messages
  - multi-collisions
  - herding attack, ...
What can happen if you don’t have a good bound?

- Length extension property
  - MAC function $h(K|M)$ not secure against forgery
  - fixing requires adding expensive construction: HMAC

- Attacks with less complexity than expected
  - 2nd pre-image for long messages
  - multi-collisions
  - herding attack, ...

- Affect all old-style hash standards: MD5, SHA-1 and all SHA-2
Hashing, scope of this SoK paper

template generation

\( Z \leftarrow T(|M|, \text{params}) \)

template execution \( H \leftarrow F(S_{\text{final}}) \) with \( S \leftarrow Y[F](Z, M) \)
Hashing, scope of this SoK paper

Template generation
\[ Z \leftarrow T(|M|, \text{params}) \]

Template execution
\[ H \leftarrow \mathcal{F}(S_{\text{final}}) \]
with
\[ S \leftarrow \mathcal{Y}[\mathcal{F}](Z, M) \]

▶ Modes \( T \) for any tree topology, including sequential hashing
Hashing, scope of this SoK paper

Template generation
\( Z \leftarrow T(|M|, \text{params}) \)

- Modes \( T \) for any tree topology, including sequential hashing
- Three types of underlying function \( F \):

Template execution
\( H \leftarrow F(S_{\text{final}}) \)
with \( S \leftarrow \mathcal{Y}[F](Z, M) \)
Hashing, scope of this SoK paper

template generation $Z \leftarrow \mathcal{T}(|M|, \text{params})$

template execution $H \leftarrow \mathcal{F}(S_{\text{final}})$ with $S \leftarrow \mathcal{Y}[\mathcal{F}](Z, M)$

- Modes $\mathcal{T}$ for any tree topology, including sequential hashing
- Three types of underlying function $\mathcal{F}$:
  - arbitrary function: XOF, hash, or compression function
Hashing, scope of this SoK paper

template generation
\[ Z \leftarrow T(|M|, \text{params}) \]

- Modes \( T \) for any tree topology, including sequential hashing
- Three types of underlying function \( \mathcal{F} \):
  - arbitrary function: XOF, hash, or compression function
  - truncated permutation

template execution \( H \leftarrow \mathcal{F}(S_{\text{final}}) \)
with \( S \leftarrow \mathcal{Y}[\mathcal{F}](Z, M) \)

message of 21 bits

\[
\begin{align*}
M_{18..20} &= 10^* 00 \\
M_{12..17} &= 00 \\
M_{6..11} &= 00 \\
M_{0..5} &= 00
\end{align*}
\]
Hashing, scope of this SoK paper

Template generation: $Z \leftarrow \mathcal{T}(|M|, \text{params})$

Template execution: $H \leftarrow \mathcal{F}(S_{\text{final}})$ with $S \leftarrow \mathcal{Y}[\mathcal{F}](Z, M)$

- Modes $\mathcal{T}$ for any tree topology, including sequential hashing
- Three types of underlying function $\mathcal{F}$:
  - arbitrary function: XOF, hash, or compression function
  - truncated permutation
  - (truncated) block cipher
Conditions for sound hashing

We prove it is hard to distinguish $H$ from $RO$ if $T$ satisfies certain conditions:

▶ For all cases:
  - message-decodability
  - subtree-freeness
  - radical-decodability

▶ For permutations and block ciphers:
  - leaf-anchoring
We prove it is hard to distinguish $\mathcal{H}$ from $\mathcal{RO}$ if $T$ satisfies certain conditions:
We prove it is hard to distinguish $H$ from $RO$ if $T$ satisfies certain conditions:

- For all cases:
  - message-decodability
  - subtree-freeness
  - radical-decodability

- For permutations and block ciphers:
  - leaf-anchoring
Trees and the set $S_T$

$S_T$: the set of all possible trees that can be generated by mode $T$
Condition 1: message decodability

There exists an algorithm for decoding \((M; Z)\) to \((M_{18..20} 00)\), \((M_{12..17} 00)\), \((M_{0..11} 00)\), and \((M_{0..5} 00)\).
Condition 1: message decodability

∀S ∈ ST there exists an algorithm for decoding S to (M, Z)
Condition 2: subtree-freeness
Condition 2: subtree-freeness

final subtree

leaf subtree

just a subtree
Condition 2: subtree-freeness
Condition 2: subtree-freeness

just a subtree

leaf subtree

final subtree
Condition 2: subtree-freeness

$$S_T$$
Condition 2: subtree-freeness

$S_{\text{sub}}$: the set of all trees that are proper subtrees of a tree in $S_T$
Condition 2: subtree-freeness

$S_{\text{sub}}$: the set of all trees that are proper subtrees of a tree in $S_T$

Subtree-freeness: $S_T \cap S_{\text{sub}} = \emptyset$
Condition 3: radical-decodability

Radical: a CV that has no \(-F\)-pre-image.
Condition 3: radical-decodability

Radical: a CV that has no $\mathcal{F}$-pre-image
Condition 3: radical-decodability

Radical-decodability, simplified: for all final subtrees ($S_{\text{final}}$) one can unambiguously identify a radical. Radical-decodability, actually: this is true for all subtrees in some set $S_T$ that includes $S_{\text{final}}$. 

Diagram:

- $S_{\text{sub}}^T$
- $S_T$
Condition 3: radical-decodability

Radical-decodability, simplified: for all final subtrees \( S_{final} \) one can unambiguously identify a radical. Radical-decodability, actually: this is true for all subtrees in some set \( S_{rad} \) that includes \( S_{final} \).
Condition 3: radical-decodability

Radical-decodability, simplified: for all final subtrees \((S_{\text{final}})\) one can unambiguously identify a radical. Radical-decodability, actually: this is true for all subtrees in some set \(S_{\text{rad}}\) that includes \(S_{\text{final}}\).
Radical-decodability, simplified: for all final subtrees \( S_{\text{final}} \) one can unambiguously identify a radical
Condition 3: radical-decodability

Radical-decodability, simplified: for all final subtrees ($S_T^{\text{final}}$) one can **unambiguously identify** a radical.

Radical-decodability, actually: this is true for all subtrees in some set $S_T^{\text{rad}}$ that includes $S_T^{\text{final}}$. 
Adversary model: differentiating from a random oracle

Indifferentiability [Maurer et al. 2004]
Adversary model: differentiating from a random oracle

Indifferentiability [Maurer et al. 2004] for hashing [Coron et al. 2005]
Adversary model: differentiating from a random oracle

- Indifferentiability [Maurer et al. 2004] for hashing [Coron et al. 2005]
- For sponge: [KT 2008] $\text{adv} \leq \binom{N}{2}2^{-c}$: birthday bound in capacity
Adversary model: differentiating from a random oracle

- Indifferentiability [Maurer et al. 2004] for hashing [Coron et al. 2005]
- For sponge: [KT 2008] $\text{adv} \leq \binom{N}{2}2^{-c}$: birthday bound in capacity
- This paper: $\text{adv} \leq \binom{N}{2}2^{-n}$: birthday bound in CV length
Adversary model: differentiating from a random oracle

Indifferentiability [Maurer et al. 2004] for hashing [Coron et al. 2005]

For sponge: [KT 2008] \( \text{adv} \leq \binom{N}{2} 2^{-c} \): birthday bound in capacity

This paper: \( \text{adv} \leq \binom{N}{2} 2^{-n} \): birthday bound in CV length

If mode satisfies our conditions
Condition 4: leaf-anchoring

- Problem with truncated permutation: inverse queries
Condition 4: leaf-anchoring

- Problem with truncated permutation: inverse queries
- Without additional condition this is easy to distinguish
Problem with truncated permutation: inverse queries
Without additional condition this is easy to distinguish
Leaf anchoring
  - \( n \) first bits of permutation input are reserved
Condition 4: leaf-anchoring

- Problem with truncated permutation: inverse queries
- Without additional condition this is easy to distinguish
- Leaf anchoring
  - $n$ first bits of permutation input are reserved
  - constant IV in leaf nodes
  - CV in non-leaf nodes
Condition 4: leaf-anchoring

- Problem with truncated permutation: inverse queries
- Without additional condition this is easy to distinguish
- Leaf anchoring
  - $n$ first bits of permutation input are reserved
  - constant IV in leaf nodes
  - CV in non-leaf nodes
- For block ciphers: anchoring must be in data input
Condition 4: leaf-anchoring

- Problem with truncated permutation: inverse queries
- Without additional condition this is easy to distinguish
- Leaf anchoring
  - $n$ first bits of permutation input are reserved
  - constant IV in leaf nodes
  - CV in non-leaf nodes
- For block ciphers: anchoring must be in data input
- Other countermeasures could be taken but this is the simplest
Problem with truncated permutation: inverse queries
Without additional condition this is easy to distinguish

Leaf anchoring
- $n$ first bits of permutation input are reserved
- constant IV in leaf nodes
- CV in non-leaf nodes

For block ciphers: anchoring must be in data input
Other countermeasures could be taken but this is the simplest
Adding a feedforward à la Davies-Meyer does not help
Minimum solutions for sequential hashing

With a compression function:

\[
\begin{array}{c}
\text{00} \\
\downarrow \\
\text{10} \\
\downarrow \\
\text{10} \\
\downarrow \\
\text{10* 11} \\
\rightarrow h
\end{array}
\]
Minimum solutions for sequential hashing

With a compression function:

With a truncated permutation or block cipher:
Interesting implications of this work

- Tree hashing mode on top of a secure XOF
Interesting implications of this work

Tree hashing mode on top of a secure XOF gives a secure XOF
Interesting implications of this work

Tree hashing mode on top of a secure XOF gives a secure XOF
- e.g., KangarooTwelve on top of sponge
Interesting implications of this work

- Tree hashing mode on top of a secure XOF gives a secure XOF
  - e.g., KangarooTwelve on top of sponge
  - Sakura encoding [KT 2014] ensures subtree-freeness and radical decodability
Interesting implications of this work

- Tree hashing mode on top of a secure XOF gives a secure XOF
  - e.g., KangarooTwelve on top of sponge
  - Sakura encoding [KT 2014] ensures subtree-freeness and radical decodability

- Hashing based on permutations
Interesting implications of this work

- Tree hashing mode on top of a secure XOF gives a secure XOF
  - e.g., KangarooTwelve on top of sponge
  - Sakura encoding [KT 2014] ensures subtree-freeness and radical decodability
- Hashing based on permutations
  - Sponge is not covered: different type of animal
Interesting implications of this work

- Tree hashing mode on top of a secure XOF gives a secure XOF
  - e.g., KangarooTwelve on top of sponge
  - Sakura encoding [KT 2014] ensures subtree-freeness and radical decodability
- Hashing based on permutations
  - Sponge is not covered: different type of animal
  - MD6: $n$-bit IV in leaves and 1 framebit would have sufficed
Interesting implications of this work

- Tree hashing mode on top of a secure XOF gives a secure XOF
  - e.g., KangarooTwelve on top of sponge
  - Sakura encoding \([KT \ 2014]\) ensures subtree-freeness and radical decodability

- Hashing based on permutations
  - Sponge is not covered: different type of animal
  - MD6: \(n\)-bit IV in leaves and 1 framebit would have sufficed

- Hashing based on block ciphers (e.g., MD5, SHA-1 and SHA-2)
  - Davies-Meyer feedforward is useless
Interesting implications of this work

- Tree hashing mode on top of a secure XOF gives a secure XOF
  - e.g., KangarooTwelve on top of sponge
  - Sakura encoding [KT 2014] ensures subtree-freeness and radical decodability

- Hashing based on permutations
  - Sponge is not covered: different type of animal
  - MD6: \( n \)-bit IV in leaves and 1 framebit would have sufficed

- Hashing based on block ciphers (e.g., MD5, SHA-1 and SHA-2)
  - Davies-Meyer feedforward is useless
  - Merkle-Damgård strengthening is useless
Interesting implications of this work

- Tree hashing mode on top of a secure XOF gives a secure XOF
  - e.g., KangarooTwelve on top of sponge
  - Sakura encoding \cite{KT2014} ensures subtree-freeness and radical decodability

- Hashing based on permutations
  - Sponge is not covered: different type of animal
  - MD6: $n$-bit IV in leaves and 1 framebit would have sufficed

- Hashing based on block ciphers (e.g., MD5, SHA-1 and SHA-2)
  - Davies-Meyer feedforward is useless
  - Merkle-Damgård strengthening is useless
  - CV can be shorter than block length of cipher
Thanks for your attention!
Intuition: why this works

- $(\mathcal{RO}, S)$ must act mode-consistent and it can:
  - Subtree-freeness $\rightarrow A$ can’t learn CVs from $(M, Z)$ queries
  - Radical-decodability $\rightarrow S$ can reconstruct any full tree $S$ queried
  - Message-decodability $\rightarrow S$ can reconstruct $M$ and $Z$ from $S$
  - $S$ then just queries $\mathcal{RO}$ with $(M, Z)$ and forwards response to $A$
Intuition: why this works

- $(RO, S)$ must act mode-consistent and it can:
  - Subtree-freeness $\rightarrow A$ can’t learn CVs from $(M, Z)$ queries
  - Radical-decodability $\rightarrow S$ can reconstruct any full tree $S$ queried
  - Message-decodability $\rightarrow S$ can reconstruct $M$ and $Z$ from $S$
  - $S$ then just queries $RO$ with $(M, Z)$ and forwards response to $A$
- Things break down when CVs collide
An example that is not radical-decodable