An Implementation of the Hash-Chain Signature Scheme for Wireless Sensor Nodes

Nadia Mourier, Reinhard Stampp, Falko Strenzke

FlexSecure GmbH, Germany,
nadia.mourier@gmail.com
reinhard.stampp@cased.de
strenzke@flexsecure.de

May 4, 2013
lightweight public key signature schemes
  - more efficient handling of keys
  - usually not lightweight
CANS 2009: Dahmen-Krauß Hash-Chain Signature scheme
Implementation of the scheme on MSP430 CPU
Correction of an error in the PRNG specification in the original paper
A new time-memory tradeoff
Evaluation of two block ciphers
Introduction

- lightweight public key signature schemes
  - more efficient handling of keys
  - usually not lightweight
- CANS 2009: Dahmen-Krauβ Hash-Chain Signature scheme
- Implementation of the scheme on MSP430 CPU
- Correction of an error in the PRNG specification in the original paper
- A new time-memory tradeoff
- Evaluation of two block ciphers
Introduction

- lightweight public key signature schemes
  - more efficient handling of keys
  - usually not lightweight
- CANS 2009: Dahmen-Krauß Hash-Chain Signature scheme
- Implementation of the scheme on MSP430 CPU
- Correction of an error in the PRNG specification in the original paper
- A new time-memory tradeoff
- Evaluation of two block ciphers
Introduction

- Lightweight public key signature schemes
  - More efficient handling of keys
  - Usually not lightweight

- CANS 2009: Dahmen-Krauß Hash-Chain Signature scheme
  - Implementation of the scheme on MSP430 CPU
  - Correction of an error in the PRNG specification in the original paper
  - A new time-memory tradeoff
  - Evaluation of two block ciphers
Introduction

- Lightweight public key signature schemes
  - More efficient handling of keys
  - Usually not lightweight

- CANS 2009: Dahmen-Krauß Hash-Chain Signature scheme
- Implementation of the scheme on MSP430 CPU
- Correction of an error in the PRNG specification in the original paper
- A new time-memory tradeoff
- Evaluation of two block ciphers
lightweight public key signature schemes
  - more efficient handling of keys
  - usually not lightweight

CANS 2009: Dahmen-Krauß Hash-Chain Signature scheme
Implementation of the scheme on MSP430 CPU
Correction of an error in the PRNG specification in the original paper
  - A new time-memory tradeoff
  - Evaluation of two block ciphers
Introduction

- lightweight public key signature schemes
  - more efficient handling of keys
  - usually not lightweight
- CANS 2009: Dahmen-Krauß Hash-Chain Signature scheme
- Implementation of the scheme on MSP430 CPU
- Correction of an error in the PRNG specification in the original paper
- A new time-memory tradeoff
- Evaluation of two block ciphers
lightweight public key signature schemes
  - more efficient handling of keys
  - usually not lightweight
CANS 2009: Dahmen-Krauß Hash-Chain Signature scheme
Implementation of the scheme on MSP430 CPU
Correction of an error in the PRNG specification in the original paper
A new time-memory tradeoff
Evaluation of two block ciphers
Outline

1. Introduction
2. Preliminaries
3. DKSS
4. Implementation
5. Conclusion
1. Introduction

2. Preliminaries

3. DKSS

4. Implementation

5. Conclusion
One-Time Signature Schemes – Key Generation

![Diagram showing key generation process]

- Verification keys: $v_1$, $v_2$, ...
- Signature keys: $s_1$, $s_2$

Mourier, Stampp, Strenzke: Implementation of Hash-Chain Signatures
One-Time Signature Schemes – Key Generation

verification keys

\[ v_1 \]

\[ H^{256}(\cdot) \]

\[ v_2 \]

signature keys

\[ s_1 \]

\[ s_2 \]

\[ \ldots \]
One-Time Signature Schemes – Key Generation

- Verification keys: $v_1$, $v_2$, ...
- Signature keys: $s_1$, $s_2$

$H^{256}()$
One-Time Signature Scheme – Signing and Verification

(\sigma_1, v_1) \rightarrow H^{256-m_1}() \rightarrow \sigma_2 \rightarrow H^{m_2}() \rightarrow (\sigma_2, v_2)

... attacker can forge signatures for \( m_i' > m_i \)

thus also a checksum must be signed:

\[ c = \sum (256 - m_i) \]
One-Time Signature Scheme – Signing and Verification

\[ \sigma_1 \xrightarrow{v_1} H^{256-m_1}(\cdot) \]
\[ \sigma_2 \xrightarrow{v_2} H^{256-m_2}(\cdot) \]

attacker can forge signatures for \( m'_i > m_i \)

thus also a checksum must be signed:

\[ c = \sum (256 - m_i) \]
One-Time Signature Scheme – Signing and Verification

$v_1\leftarrow H^{256-m_1()}$
$s_1\leftarrow v_1$
$s_2\leftarrow H^{m_2()}$
$v_2\leftarrow H^{256-m_2()}$

... attacker can forge signatures for $m'_i > m_i$
thus also a checksum must be signed:

\[ c = \sum (256 - m_i) \]
One-Time Signature Scheme – Signing and Verification

\[ v_1 \]
\[ H^{256-m_1}() \]
\[ \sigma_1 \]
\[ H^{m_1}() \]
\[ s_1 \]

\[ v_2 \]
\[ H^{256-m_2}() \]
\[ \sigma_2 \]
\[ H^{m_2}() \]
\[ s_2 \]

... attacker can forge signatures for \( m'_i > m_i \)
thus also a checksum must be signed:

\[ c = \sum (256 - m_i) \]
One-Time Signature Scheme – Signing and Verification

\[ \sigma_1, v_1 \]

\[ H^{256 - m_1}() \]

\[ s_1 \]

\[ H^{m_1}() \]

\[ H^{256 - m_2}() \]

\[ v_2 \]

\[ \sigma_2 \]

\[ H^{m_2}() \]

\[ s_2 \]

Attacker can forge signatures for \( m'_i > m_i \)

Thus also a checksum must be signed:

\[ c = \sum (256 - m_i) \]

Mourier, Stampp, Strenzke: Implementation of Hash-Chain Signatures
One-Time Signature Scheme – Signing and Verification

\[ \sigma_1 \]
\[ H^{m_1}() \]
\[ s_1 \]

\[ \sigma_2 \]
\[ H^{m_2}() \]
\[ s_2 \]

\[ v_1 \]
\[ H^{256-m_1}() \]

\[ v_2 \]
\[ H^{256-m_2}() \]

\[ \ldots \]

attacker can forge signatures for \( m_i' > m_i \)

thus also a checksum must be signed:

\[ c = \sum (256 - m_i) \]
One-Time Signature Scheme – Signing and Verification

$\sigma_1$ $\triangleleft$ $H^{256-m_1}()$

$v_1$

$H^{m_1}()$

$s_1$

$\sigma_2$

$v_2$

$H^{256-m_2}()$

$s_2$

attacker can forge signatures for $m_i' > m_i$

thus also a checksum must be signed:

$c = \sum (256 - m_i)$
One-Time Signature Scheme – Signing and Verification

$v_1 \quad H^{256-m_1}() \\
\sigma_1 \quad H^{m_1}() \quad S_1$

$v_2 \quad H^{256-m_2}() \\
\sigma_2 \quad H^{m_2}() \quad S_2$

... attacker can forge signatures for $m'_i > m_i$
thus also a checksum must be signed:

\[ c = \sum (256 - m_i) \]
One-Time Signature Scheme – Signing and Verification

\[ H^{256-m_1}() \]

\[ H^{256-m_2}() \]

\[ c = \sum (256 - m_i) \]

attacker can forge signatures for \( m'_i > m_i \)

thus also a checksum must be signed:
One-Time Signature Scheme – Signing and Verification

\[ H_{256-m_1}() \]

\[ H_{256-m_2}() \]

\[ s_1 \]

\[ s_2 \]

\[ v_1 \]

\[ v_2 \]

\[ \sigma_1 \]

\[ \sigma_2 \]

attacker can forge signatures for \( m'_i > m_i \)

thus also a checksum must be signed:

\[ c = \sum (256 - m_i) \]
One-Time Signature Scheme – Signing and Verification

\[ v_1 \]
\[ H^{256-m_1()} \]
\[ \sigma_1 \]
\[ H^{m_1()} \]
\[ s_1 \]

\[ v_2 \]
\[ H^{256-m_2()} \]
\[ \sigma_2 \]
\[ H^{m_2()} \]
\[ s_2 \]

\[ c = \sum (256 - m_i) \]

... attacker can forge signatures for \( m'_i > m_i \) 
thus also a checksum must be signed:
One-Time Signature Scheme – Signing and Verification

\[ \sigma_1 \quad H^{m_1}() \quad s_1 \]

\[ \sigma_2 \quad H^{m_2}() \quad s_2 \]

\[ \forall \quad H^{256-m_1()} \quad v_1 \]

\[ \forall \quad H^{256-m_2()} \quad v_2 \]

\[ c = \sum (256 - m_i) \]

attacker can forge signatures for \( m'_i > m_i \)

thus also a checksum must be signed:
Multiple Signature Schemes from One-Time Signature Schemes

- efficient handling of signature keys:
  - on demand creation through PRNG
  - small public key
  - Merkle Tree Scheme
  - Hash Chain Signature Scheme by Dahmen and Krauß
Multiple Signature Schemes from One-Time Signature Schemes

- efficient handling of signature keys:
  - on demand creation through PRNG
- small public key
  - Merkle Tree Scheme
  - Hash Chain Signature Scheme by Dahmen and Krauβ
Multiple Signature Schemes from One-Time Signature Schemes

- efficient handling of signature keys:
  - on demand creation through PRNG
- small public key
  - Merkle Tree Scheme
  - Hash Chain Signature Scheme by Dahmen and Krauß
Multiple Signature Schemes from One-Time Signature Schemes

- efficient handling of signature keys:
  - on demand creation through PRNG
- small public key
  - Merkle Tree Scheme
  - Hash Chain Signature Scheme by Dahmen and Krauß
Multiple Signature Schemes from One-Time Signature Schemes

- efficient handling of signature keys:
  - on demand creation through PRNG
- small public key
  - Merkle Tree Scheme
- Hash Chain Signature Scheme by Dahmen and Krauß
DKSS Key Generation

signature generation

\[ Z_0 \leftarrow Z_1 \leftarrow Z_2 \leftarrow \cdots \leftarrow Z_{l-1} \leftarrow Z_l \]
\[ Y_1 \uparrow \quad Y_2 \uparrow \quad \cdots \uparrow \quad Y_{l-1} \uparrow \quad Y_l \]
\[ PRNG \rightarrow X_1 \quad X_2 \quad \cdots \quad X_{l-1} \quad X_l \]

Hash chain traversal algorithm: Yum et al., CT-RSA 2009:
Storage: \([1/2 \log l]\) links \(z_i\)
Computational cost: \([\log l]\)
DKSS Key Generation

Hash chain traversal algorithm: Yum et al., CT-RSA 2009:
Storage: \( \lceil \frac{1}{2} \log l \rceil \) links \( z_i \)
Computational cost: \( \lceil \log l \rceil \)
DKSS Key Generation

signature generation

\[ \begin{align*}
Z_0 & \leftarrow Z_1 \leftarrow Z_2 \leftarrow \cdots \leftarrow Z_{l-1} \leftarrow Z_l \\
Y_1 & \leftarrow Y_2 \leftarrow \cdots \leftarrow Z_{l-1} \leftarrow Z_l \\
X_1 & \leftarrow X_2 \leftarrow \cdots \leftarrow Z_{l-1} \leftarrow Z_l
\end{align*} \]

Hash chain traversal algorithm: Yum et al., CT-RSA 2009:
Storage: \( \lceil 1/2 \log l \rceil \) links \( z_i \)
Computational cost: \( \lceil \log l \rceil \)
DKSS Key Generation

signature generation

\[ Z_0 \leftarrow Z_1 \leftarrow Z_2 \leftarrow \cdots \leftarrow Z_{l-1} \leftarrow Z_l \]

\[ Y_1 \leftarrow Y_2 \leftarrow \cdots \leftarrow Y_{l-1} \leftarrow Y_l \]

\[ X_1 \leftarrow X_2 \leftarrow \cdots \leftarrow X_{l-1} \leftarrow X_l \]

Hash chain traversal algorithm: Yum et al., CT-RSA 2009:

Storage: \( \lceil \frac{1}{2} \log l \rceil \) links \( z_i \)

Computational cost: \( \lceil \log l \rceil \)
DKSS Key Generation

signature generation

\[ Z_0 \leftarrow Z_1 \leftarrow Z_2 \leftarrow \cdots \leftarrow Z_{l-1} \leftarrow Z_l \]

\[ Y_1 \leftarrow Y_2 \leftarrow \cdots \leftarrow Y_l \]

\[ X_1 \leftarrow X_2 \leftarrow \cdots \leftarrow X_l \]

PRNG \rightarrow

Hash chain traversal algorithm: Yum et al., CT-RSA 2009:

Storage: \( \lceil 1/2 \log l \rceil \) links \( z_i \)

Computational cost: \( \lceil \log l \rceil \)
DKSS Key Generation

signature generation

\[ \begin{align*}
Z_0 & \leftarrow Z_1 & \cdots & \leftarrow Z_{l-1} & \leftarrow Z_l \\
Y_1 & \leftarrow Y_2 & \cdots & \leftarrow Y_{l-1} & \leftarrow Y_l \\
X_1 & \leftarrow X_2 & \cdots & \leftarrow X_{l-1} & \leftarrow X_l \\
\end{align*} \]

Hash chain traversal algorithm: Yum et al., CT-RSA 2009:
Storage: \( \lceil \frac{1}{2} \log l \rceil \) links \( z_i \)
Computational cost: \( \lceil \log l \rceil \)
DKSS Key Generation

signature generation

\[ Z_0 \leftarrow Z_1 \leftarrow Z_2 \leftarrow \cdots \leftarrow Z_{l-1} \leftarrow Z_l \]

\[ Y_1 \leftarrow Y_2 \leftarrow \cdots \leftarrow Y_{l-1} \leftarrow Y_l \]

\[ X_1 \leftarrow X_2 \leftarrow \cdots \leftarrow X_{l-1} \leftarrow X_l \]

PRNG\(\rightarrow\) Hash chain traversal algorithm: Yum et al., CT-RSA 2009:

- Storage: \(\lceil 1/2 \log l \rceil\) links \(z_i\)
- Computational cost: \(\lceil \log l \rceil\)
Parameters and Hash Functions of the Scheme

- \( w \) – bit size of message
- \( l \) – number of signatures
- \( n \) – security level (bit size of hash output)

hash functions
- \( f : \{0,1\}^n \to \{0,1\}^n \)
- \( g : \{0,1\}^{4n} \to \{0,1\}^n \)

\[
f(x) = \lfloor E_{IV}(x) \oplus x \rfloor_n
\]
\[
g(x_1, x_2, x_3, x_4) = \lfloor E_{k_3}(x_4) \oplus x_4 \rfloor_n
\]
with
\[
k_3 = E_{k_2}(x_3) \oplus x_3
\]
\[
k_2 = E_{k_1}(x_2) \oplus x_2
\]
\[
k_1 = E_{IV}(x_1) \oplus x_1
\]
Parameters and Hash Functions of the Scheme

- $w$ – bit size of message
- $l$ – number of signatures
- $n$ – security level (bit size of hash output)

Hash functions

- $f : \{0,1\}^n \rightarrow \{0,1\}^n$
- $g : \{0,1\}^{4n} \rightarrow \{0,1\}^n$

$$f(x) = [E_{IV}(x) \oplus x]_n$$

$$g(x_1, x_2, x_3, x_4) = [E_{k_3}(x_4) \oplus x_4]_n$$

with $k_3 = E_{k_2}(x_3) \oplus x_3$

$k_2 = E_{k_1}(x_2) \oplus x_2$

$k_1 = E_{IV}(x_1) \oplus x_1$
Parameters and Hash Functions of the Scheme

- $w$ – bit size of message
- $l$ – number of signatures
- $n$ – security level (bit size of hash output)

Hash functions

- $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$
- $g : \{0, 1\}^{4n} \rightarrow \{0, 1\}^n$

\[
f(x) = \lfloor E_{IV}(x) \oplus x \rfloor_n\]
\[
g(x_1, x_2, x_3, x_4) = \lfloor E_{k_3}(x_4) \oplus x_4 \rfloor_n\]

with
\[
k_3 = E_{k_2}(x_3) \oplus x_3\]
\[
k_2 = E_{k_1}(x_2) \oplus x_2\]
\[
k_1 = E_{IV}(x_1) \oplus x_1\]
Parameters and Hash Functions of the Scheme

- $w$ – bit size of message
- $l$ – number of signatures
- $n$ – security level (bit size of hash output)

hash functions

$$f : \{0, 1\}^n \rightarrow \{0, 1\}^n$$
$$g : \{0, 1\}^{4n} \rightarrow \{0, 1\}^n$$

$$f(x) = \lfloor E_{IV}(x) \oplus x \rfloor_n$$
$$g(x_1, x_2, x_3, x_4) = \lfloor E_{k_3}(x_4) \oplus x_4 \rfloor_n$$

with $k_3 = E_{k_2}(x_3) \oplus x_3$
$$k_2 = E_{k_1}(x_2) \oplus x_2$$
$$k_1 = E_{IV}(x_1) \oplus x_1$$
Parameters and Hash Functions of the Scheme

- \( w \) – bit size of message
- \( l \) – number of signatures
- \( n \) – security level (bit size of hash output)

hash functions

- \( f : \{0, 1\}^n \rightarrow \{0, 1\}^n \)
- \( g : \{0, 1\}^{4n} \rightarrow \{0, 1\}^n \)

\[
f(x) = [E_{IV}(x) \oplus x]_n
\]

\[
g(x_1, x_2, x_3, x_4) = [E_{k_3}(x_4) \oplus x_4]_n
\]

with

\[
k_3 = E_{k_2}(x_3) \oplus x_3
\]

\[
k_2 = E_{k_1}(x_2) \oplus x_2
\]

\[
k_1 = E_{IV}(x_1) \oplus x_1
\]
Parameters and Hash Functions of the Scheme

- \( w \) – bit size of message
- \( l \) – number of signatures
- \( n \) – security level (bit size of hash output)

Hash functions

- \( f : \{0, 1\}^n \rightarrow \{0, 1\}^n \)
- \( g : \{0, 1\}^{4n} \rightarrow \{0, 1\}^n \)

\[
f(x) = \lfloor E_{IV}(x) \oplus x \rfloor_n
\]

\[
g(x_1, x_2, x_3, x_4) = \lfloor E_{k_3}(x_4) \oplus x_4 \rfloor_n
\]

with

\[
k_3 = E_{k_2}(x_3) \oplus x_3
\]

\[
k_2 = E_{k_1}(x_2) \oplus x_2
\]

\[
k_1 = E_{IV}(x_1) \oplus x_1
\]
Parameters and Hash Functions of the Scheme

- \( w \) – bit size of message
- \( l \) – number of signatures
- \( n \) – security level (bit size of hash output)

hash functions

- \( f : \{0, 1\}^n \rightarrow \{0, 1\}^n \)
- \( g : \{0, 1\}^{4n} \rightarrow \{0, 1\}^n \)

\[
f(x) = \lfloor E_{IV}(x) \oplus x \rfloor_n \quad g(x_1, x_2, x_3, x_4) = \lfloor E_{k_3}(x_4) \oplus x_4 \rfloor_n
\]

with
\[
k_3 = E_{k_2}(x_3) \oplus x_3
k_2 = E_{k_1}(x_2) \oplus x_2
k_1 = E_{IV}(x_1) \oplus x_1
\]
Signature Size

- security level $n = 80$
- signature size for $w = 16$: 336 bits
- ECDSA 160: 320 bits
- signing of short fixed size messages ($\neq$ Merkle)
- Verifier need all previous signatures ($\neq$ Merkle)
- fixed number of signatures per public key ($=\text{Merkle}$)
- small public key (80 bit) ($=\text{Merkle}$)
- verification much faster than signature generation
- intended application: broadcast messages in WSNs
Features

Signature Size

- security level $n = 80$
- signature size for $w = 16$: 336 bits
- ECDSA 160: 320 bits

- signing of short fixed size messages ($\neq$ Merkle)
- Verifier need all previous signatures ($\neq$ Merkle)
- fixed number of signatures per public key ($= \text{Merkle}$)
- small public key (80 bit) ($= \text{Merkle}$)
- verification much faster than signature generation
- intended application: broadcast messages in WSNs
Features

- **Signature Size**
  - security level $n = 80$
  - signature size for $w = 16$: 336 bits
  - ECDSA 160: 320 bits

- signing of short fixed size messages ($\neq$ Merkle)
- Verifier need all previous signatures ($\neq$ Merkle)
- fixed number of signatures per public key ($= \text{Merkle}$)
- small public key (80 bit) ($= \text{Merkle}$)
- verification much faster than signature generation
- intended application: broadcast messages in WSNs

Mourier, Stampp, Strenzke: Implementation of Hash-Chain Signatures
Features

- **Signature Size**
  - security level $n = 80$
  - signature size for $w = 16$: 336 bits
  - ECDSA 160: 320 bits

- signing of short fixed size messages ($\neq$ Merkle)
- Verifier need all previous signatures ($\neq$ Merkle)
- fixed number of signatures per public key ($=$ Merkle)
- small public key (80 bit) ($=$ Merkle)
- verification much faster than signature generation
- intended application: broadcast messages in WSNs
Features

- **Signature Size**
  - security level $n = 80$
  - signature size for $w = 16$: 336 bits
  - ECDSA 160: 320 bits

- signing of short fixed size messages ($\neq$ Merkle)
  - Verifier need all previous signatures ($\neq$ Merkle)
  - fixed number of signatures per public key ($=$ Merkle)
  - small public key (80 bit) ($=$ Merkle)
  - verification much faster than signature generation
  - intended application: broadcast messages in WSNs
Features

- **Signature Size**
  - security level $n = 80$
  - signature size for $w = 16$: 336 bits
  - ECDSA 160: 320 bits

- signing of short fixed size messages ($\neq$ Merkle)
- Verifier need all previous signatures ($\neq$ Merkle)
  - fixed number of signatures per public key ($\equiv$ Merkle)
  - small public key (80 bit) ($\equiv$ Merkle)
  - verification much faster than signature generation
- intended application: broadcast messages in WSNs

Mourier, Stampp, Strenzke: Implementation of Hash-Chain Signatures
Features

- Signature Size
  - security level $n = 80$
  - signature size for $w = 16$: 336 bits
  - ECDSA 160: 320 bits
- signing of short fixed size messages (≠ Merkle)
- Verifier need all previous signatures (≠ Merkle)
- fixed number of signatures per public key (= Merkle)
- small public key (80 bit) (= Merkle)
- verification much faster than signature generation
- intended application: broadcast messages in WSNs
Features

- **Signature Size**
  - security level $n = 80$
  - signature size for $w = 16$: 336 bits
  - ECDSA 160: 320 bits

- signing of short fixed size messages ($\neq$ Merkle)
- Verifier need all previous signatures ($\neq$ Merkle)
- fixed number of signatures per public key ($=\text{Merkle}$)
- small public key (80 bit) ($=\text{Merkle}$)
- verification much faster than signature generation
- intended application: broadcast messages in WSNs
Features

- **Signature Size**
  - security level $n = 80$
  - signature size for $w = 16$: 336 bits
  - ECDSA 160: 320 bits
- signing of short fixed size messages ($\neq$ Merkle)
- Verifier need all previous signatures ($\neq$ Merkle)
- fixed number of signatures per public key ($=\text{Merkle}$)
- small public key (80 bit) ($=\text{Merkle}$)
- verification much faster than signature generation
- intended application: broadcast messages in WSNs
Features

- **Signature Size**
  - security level $n = 80$
  - signature size for $w = 16$: 336 bits
  - ECDSA 160: 320 bits

- signing of short fixed size messages ($\neq$ Merkle)
- Verifier need all previous signatures ($\neq$ Merkle)
- fixed number of signatures per public key ($=\text{ Merkle}$)
- small public key (80 bit) ($=\text{ Merkle}$)
- verification much faster than signature generation
- intended application: broadcast messages in WSNs
Details of DKSS Key Generation

- Compute the one-time signature keys
  \[ X_i = (x_i[0], x_i[1], x_i[2]) \in \{0, 1\}^{(n,3)}: \]
  - for \( i = 1 \) to \( l \)
    - \( x_i[0] \leftarrow \text{PRNG}(i, 0) \)
    - \( x_i[1] \leftarrow \text{PRNG}(i, 1) \)
    - \( x_i[2] \leftarrow \text{PRNG}(i, 2) \)

- Calculate the one-time verification key
  \[ Y_i = (y_i[0], y_i[1], y_i[2]) \in \{0, 1\}^{(n,3)}: \]
  - for \( i = l \) to \( 1 \)
    - \( y_i[0] \leftarrow f^{2^{y_i[0]} - 1}(x_i[0]) \)
    - \( y_i[1] \leftarrow f^{2^{y_i[1]} - 1}(x_i[1]) \)
    - \( y_i[2] \leftarrow f^{2^{y_i[2]} + 1} - 2(x_i[2]) \)
    - \( z_{i-1} \leftarrow g(y_i[0] \parallel y_i[1] \parallel y_i[2] \parallel z_i) \)
Details of DKSS Key Generation

- Compute the one-time signature keys
  \[ X_i = (x_i[0], x_i[1], x_i[2]) \in \{0, 1\}^{(n,3)}: \]
  - for \( i = 1 \) to \( l \)
    - \( x_i[0] \leftarrow \text{PRNG}(i, 0) \)
    - \( x_i[1] \leftarrow \text{PRNG}(i, 1) \)
    - \( x_i[2] \leftarrow \text{PRNG}(i, 2) \)

- Calculate the one-time verification key
  \[ Y_i = (y_i[0], y_i[1], y_i[2]) \in \{0, 1\}^{(n,3)}: \]
  - for \( i = l \) to \( 1 \)
    - \( y_i[0] \leftarrow f_2^{2^2 - 1}(x_i[0]) \)
    - \( y_i[1] \leftarrow f_2^{2^2 - 1}(x_i[1]) \)
    - \( y_i[2] \leftarrow f_2^{2^2 - 1}(x_i[2]) \)
    - \( z_{i-1} \leftarrow g(y_i[0] \parallel y_i[1] \parallel y_i[2] \parallel z_i) \)
Details of DKSS Key Generation

- Compute the one-time signature keys
  \[ X_i = (x_i[0], x_i[1], x_i[2]) \in \{0, 1\}^{(n,3)}: \]
  - for \(i = 1\) to \(l\)
    - \(x_i[0] \leftarrow \text{PRNG}(i, 0)\)
    - \(x_i[1] \leftarrow \text{PRNG}(i, 1)\)
    - \(x_i[2] \leftarrow \text{PRNG}(i, 2)\)

- Calculate the one-time verification key
  \[ Y_i = (y_i[0], y_i[1], y_i[2]) \in \{0, 1\}^{(n,3)}: \]
  - for \(i = l\) to 1
    - \(y_i[0] \leftarrow f^2y_i[0]^{-1}(x_i[0])\)
    - \(y_i[1] \leftarrow f^2y_i[1]^{-1}(x_i[1])\)
    - \(y_i[2] \leftarrow f^2y_i[2]^{-1}(x_i[2])\)
    - \(z_i \leftarrow g(y_i[0] \parallel y_i[1] \parallel y_i[2] \parallel z_i)\)
Compute the one-time signature keys

\[ X_i = (x_i[0], x_i[1], x_i[2]) \in \{0, 1\}^{(n,3)}: \]

for \( i = 1 \) to \( l \)
- \( x_i[0] \leftarrow \text{PRNG}(i, 0) \)
- \( x_i[1] \leftarrow \text{PRNG}(i, 1) \)
- \( x_i[2] \leftarrow \text{PRNG}(i, 2) \)

Calculate the one-time verification key

\[ Y_i = (y_i[0], y_i[1], y_i[2]) \in \{0, 1\}^{(n,3)}: \]

for \( i = l \) to \( 1 \)
- \( y_i[0] \leftarrow f^{2^{\frac{w}{2} - 1}}(x_i[0]) \)
- \( y_i[1] \leftarrow f^{2^{\frac{w}{2} - 1}}(x_i[1]) \)
- \( y_i[2] \leftarrow f^{2^{\frac{w}{2} + 1} - 2}(x_i[2]) \)
- \( z_{i-1} \leftarrow g(y_i[0] \parallel y_i[1] \parallel y_i[2] \parallel z_i) \)
Detailed of DKSS Key Generation

- Compute the one-time signature keys
  \[ X_i = (x_i[0], x_i[1], x_i[2]) \in \{0, 1\}^{(n,3)} \]
  for \( i = 1 \) to \( l \)
  \[ x_i[0] \leftarrow \text{PRNG}(i, 0) \]
  \[ x_i[1] \leftarrow \text{PRNG}(i, 1) \]
  \[ x_i[2] \leftarrow \text{PRNG}(i, 2) \]

- Calculate the one-time verification key
  \[ Y_i = (y_i[0], y_i[1], y_i[2]) \in \{0, 1\}^{(n,3)} \]
  for \( i = l \) to 1
  \[ y_i[0] \leftarrow f^{2^q - 1}(x_i[0]) \]
  \[ y_i[1] \leftarrow f^{2^q - 1}(x_i[1]) \]
  \[ y_i[2] \leftarrow f^{2^q + 1 - 2}(x_i[2]) \]
  \[ z_i \leftarrow g(y_i[0] \parallel y_i[1] \parallel y_i[2] \parallel z_i) \]
Details of DKSS Key Generation

- Compute the one-time signature keys
  \[ X_i = (x_i[0], x_i[1], x_i[2]) \in \{0, 1\}^{(n,3)}: \]
  for \( i = 1 \) to \( l \)
  - \( x_i[0] \leftarrow \text{PRNG}(i, 0) \)
  - \( x_i[1] \leftarrow \text{PRNG}(i, 1) \)
  - \( x_i[2] \leftarrow \text{PRNG}(i, 2) \)

- Calculate the one-time verification key
  \[ Y_i = (y_i[0], y_i[1], y_i[2]) \in \{0, 1\}^{(n,3)}: \]
  for \( i = l \) to \( 1 \)
  - \( y_i[0] \leftarrow f^{2^3 w^2 - 1}(x_i[0]) \)
  - \( y_i[1] \leftarrow f^{2^3 w^2 - 1}(x_i[1]) \)
  - \( y_i[2] \leftarrow f^{2^3 w^2 + 1 - 2}(x_i[2]) \)
  - \( z_i \leftarrow g(y_i[0] \parallel y_i[1] \parallel y_i[2] \parallel z_i) \)
Details of DKSS Key Generation

- Compute the one-time signature keys
  \[ X_i = (x_i[0], x_i[1], x_i[2]) \in \{0, 1\}^{(n,3)} : \]
  
  for \( i = 1 \) to \( l \)
  - \( x_i[0] \leftarrow \text{PRNG}(i, 0) \)
  - \( x_i[1] \leftarrow \text{PRNG}(i, 1) \)
  - \( x_i[2] \leftarrow \text{PRNG}(i, 2) \)

- Calculate the one-time verification key
  \[ Y_i = (y_i[0], y_i[1], y_i[2]) \in \{0, 1\}^{(n,3)} : \]
  
  for \( i = l \) to \( 1 \)
  - \( y_i[0] \leftarrow f^{2^{w_2/2} - 1}(x_i[0]) \)
  - \( y_i[1] \leftarrow f^{2^{w_2/2} - 1}(x_i[1]) \)
  - \( y_i[2] \leftarrow f^{2^{w_2/2} + 1 - 2}(x_i[2]) \)
  - \( z_i \leftarrow g(y_i[0] \parallel y_i[1] \parallel y_i[2] \parallel z_i) \)
Details of DKSS Key Generation

- Compute the one-time signature keys $X_i = (x_i[0], x_i[1], x_i[2]) \in \{0, 1\}^{(n,3)}$:
  - for $i = 1$ to $l$
    - $x_i[0] \leftarrow \text{PRNG}(i, 0)$
    - $x_i[1] \leftarrow \text{PRNG}(i, 1)$
    - $x_i[2] \leftarrow \text{PRNG}(i, 2)$

- Calculate the one-time verification key $Y_i = (y_i[0], y_i[1], y_i[2]) \in \{0, 1\}^{(n,3)}$:
  - for $i = l$ to $1$
    - $y_i[0] \leftarrow f^{2^{\frac{w}{2}} - 1}(x_i[0])$
    - $y_i[1] \leftarrow f^{2^{\frac{w}{2}} - 1}(x_i[1])$
    - $y_i[2] \leftarrow f^{2^{\frac{w}{2} + 1} - 2}(x_i[2])$
    - $z_{i-1} \leftarrow g(y_i[0] \parallel y_i[1] \parallel y_i[2] \parallel z_i)$
Details of DKSS Key Generation

- Compute the one-time signature keys
  \[ X_i = (x_i[0], x_i[1], x_i[2]) \in \{0, 1\}^{(n,3)}: \]
  - for \( i = 1 \) to \( l \)
    - \( x_i[0] \leftarrow \text{PRNG}(i, 0) \)
    - \( x_i[1] \leftarrow \text{PRNG}(i, 1) \)
    - \( x_i[2] \leftarrow \text{PRNG}(i, 2) \)

- Calculate the one-time verification key
  \[ Y_i = (y_i[0], y_i[1], y_i[2]) \in \{0, 1\}^{(n,3)}: \]
  - for \( i = l \) to 1
    - \( y_i[0] \leftarrow f^{\frac{w}{2} - 1}(x_i[0]) \)
    - \( y_i[1] \leftarrow f^{\frac{w}{2} - 1}(x_i[1]) \)
    - \( y_i[2] \leftarrow f^{\frac{w}{2} + 1 - 2}(x_i[2]) \)
    - \( z_{i-1} \leftarrow g(y_i[0] \parallel y_i[1] \parallel y_i[2] \parallel z_i) \)
Details of DKSS Key Generation

- Compute the one-time signature keys
  \( X_i = (x_i[0], x_i[1], x_i[2]) \in \{0, 1\}^{(n,3)}: \)

  for \( i = 1 \) to \( l \)
  - \( x_i[0] \leftarrow \text{PRNG}(i, 0) \)
  - \( x_i[1] \leftarrow \text{PRNG}(i, 1) \)
  - \( x_i[2] \leftarrow \text{PRNG}(i, 2) \)

- Calculate the one-time verification key
  \( Y_i = (y_i[0], y_i[1], y_i[2]) \in \{0, 1\}^{(n,3)}: \)

  for \( i = l \) to \( 1 \)
  - \( y_i[0] \leftarrow f^{2^{\frac{w}{2}} - 1}(x_i[0]) \)
  - \( y_i[1] \leftarrow f^{2^{\frac{w}{2}} - 1}(x_i[1]) \)
  - \( y_i[2] \leftarrow f^{2^{\frac{w}{2} + 1} - 2}(x_i[2]) \)
  - \( z_{i-1} \leftarrow g(y_i[0] \parallel y_i[1] \parallel y_i[2] \parallel z_i) \)
Details of DKSS Key Generation

- Compute the one-time signature keys
  \[ X_i = (x_i[0], x_i[1], x_i[2]) \in \{0, 1\}^{(n,3)}: \]
  for \( i = 1 \) to \( l \)
    - \( x_i[0] \leftarrow \text{PRNG}(i, 0) \)
    - \( x_i[1] \leftarrow \text{PRNG}(i, 1) \)
    - \( x_i[2] \leftarrow \text{PRNG}(i, 2) \)

- Calculate the one-time verification key
  \[ Y_i = (y_i[0], y_i[1], y_i[2]) \in \{0, 1\}^{(n,3)}: \]
  for \( i = l \) to \( 1 \)
    - \( y_i[0] \leftarrow f^{2^{\frac{w}{2}}-1}(x_i[0]) \)
    - \( y_i[1] \leftarrow f^{2^{\frac{w}{2}}-1}(x_i[1]) \)
    - \( y_i[2] \leftarrow f^{2^{\frac{w}{2}}+1-2}(x_i[2]) \)
    - \( z_{i-1} \leftarrow g(y_i[0] \parallel y_i[1] \parallel y_i[2] \parallel z_i) \)
DKSS Signature Generation and Verification

\[ m = m_1 \| m_2 \]

\[ y_i[0] \]  
\[ f^{m_1(x_i[0])} \]
\[ f^{2^{w/2} - 1 - m_1(\alpha_1)} \]
\[ \alpha_1 \]
\[ x_i[0] \]

\[ y_i[1] \]
\[ f^{m_2(x_i[1])} \]
\[ f^{2^{w/2} - 1 - m_2(\alpha_2)} \]
\[ \alpha_2 \]
\[ x_i[1] \]

\[ y_i[2] \]
\[ f^{2^{w/2} - 1 - 2 - c(\alpha_3)} \]
\[ \alpha_3 \]
\[ x_i[2] \]

\[ c \leftarrow 2^{w/2 + 1} - 2 - m_1 - m_2 \]

Verification:
\[ g(y_i[0] \| y_i[1] \| y_i[2] \| z_i) = z_{i-1} \]
$$m = m_1 \parallel m_2$$

$$y_i [0]$$  
$$f^{m_1} (x_i [0])$$  
$$f^{2^{w/2} - 1 - m_1} (\alpha_1)$$  
$$\alpha_1$$

$$y_i [1]$$  
$$f^{m_2} (x_i [1])$$  
$$f^{2^{w/2} - 1 - m_2} (\alpha_2)$$  
$$\alpha_2$$

$$y_i [2]$$  
$$f^{m_2} (x_i [2])$$  
$$f^{2^{w/2} + 1 - 2 - c} (\alpha_3)$$  
$$\alpha_3$$

$$x_i [0]$$  
$$f^{m_1} (x_i [0])$$

$$x_i [1]$$  
$$f^{m_2} (x_i [1])$$

$$x_i [2]$$  
$$f^{c} (x_i [2])$$

$$c \leftarrow 2^{w/2 + 1} - 2 - m_1 - m_2$$

verification:  
$$g(y_i [0] \parallel y_i [1] \parallel y_i [2] \parallel z_i) = z_{i-1}$$
m = m_1 || m_2

\[ y_i[0] \]
\[ f^{2^{w/2} - 1 - m_1}(\alpha_1) \]
\[ f^{m_1}(x_i[0]) \]
\[ x_i[0] \]
\[ \alpha_1 \]

\[ y_i[1] \]
\[ f^{2^{w/2} - 1 - m_2}(\alpha_2) \]
\[ f^{m_2}(x_i[1]) \]
\[ x_i[1] \]
\[ \alpha_2 \]

\[ y_i[2] \]
\[ f^{2^{w/2} + 1 - 2 - c}(\alpha_3) \]
\[ f^{c}(x_i[2]) \]
\[ x_i[2] \]
\[ \alpha_3 \]

\[ c \leftarrow 2^{w/2} + 1 - 2 - m_1 - m_2 \]

verification: \( g(y_i[0] || y_i[1] || y_i[2] || z_i) = z_{i-1} \)
$m = m_1 \parallel m_2$

c ← $2^{\frac{w}{2} + 1} - 2 - m_1 - m_2$

verification: $g(y_i[0] \parallel y_i[1] \parallel y_i[2] \parallel z_i) = z_{i-1}$?
DKSS Signature Generation and Verification

\[ m = m_1 \| m_2 \]

\[ y_i[0] \quad y_i[1] \quad y_i[2] \]

\[ f^{2^w - 1 - m_1}(\alpha_1) \quad \alpha_2 \quad f^{2^w + 1 - 2 - c}(\alpha_3) \]

\[ x_i[0] \quad x_i[1] \quad x_i[2] \]

\[ f^{m_1}(x_i[0]) \quad f^{m_2}(x_i[1]) \quad f^c(x_i[2]) \]

\[ c \leftarrow 2^{\frac{w}{2} + 1} - 2 - m_1 - m_2 \]

Verification:  \( g(y_i[0] \| y_i[1] \| y_i[2] \| z_i) = z_{i-1} \)
DKSS Signature Generation and Verification

\[ m = m_1 \| m_2 \]

\[ y_i[0] \quad y_i[1] \quad y_i[2] \]

\[ f^{2^{w/2} - 1 - m_1}(\alpha_1) \]

\[ f^{m_2}(x_i[0]) \]

\[ f^{m_2}(x_i[1]) \]

\[ f^{2^{w/2} + 1 - 2 - c}(\alpha_3) \]

\[ f^c(x_i[2]) \]

\[ x_i[0] \quad x_i[1] \quad x_i[2] \]

\[ c \leftarrow 2^{w/2 + 1} - 2 - m_1 - m_2 \]

Verification: \( g(y_i[0] \| y_i[1] \| y_i[2] \| z_i) = z_{i-1} \)
\[ m = m_1 \| m_2 \]

\[ y_i[0] \]
\[ y_i[1] \]
\[ y_i[2] \]

\[ x_i[0] \]
\[ x_i[1] \]
\[ x_i[2] \]

\[ c \leftarrow 2^\frac{w+1}{2} - 2 - m_1 - m_2 \]

\[
g(y_i[0] \| y_i[1] \| y_i[2] \| z_i) = z_{i-1}^?\]
DKSS Signature Generation and Verification

\[ m = m_1 \parallel m_2 \]

\[ y_i[0] \]

\[ f^{2^{w/2} - 1 - m_1}(\alpha_1) \]

\[ f^{m_1}(x_i[0]) \]

\[ x_i[0] \]

\[ c \leftarrow 2^{w/2 + 1} - 2 - m_1 - m_2 \]

\[ y_i[1] \]

\[ f^{2^{w/2} - 1 - m_2}(\alpha_2) \]

\[ f^{m_2}(x_i[1]) \]

\[ x_i[1] \]

\[ y_i[2] \]

\[ f^{2^{w/2 + 1} - 2 - c}(\alpha_3) \]

\[ f^{c}(x_i[2]) \]

\[ x_i[2] \]

\[ z_i \]

\[ c \leftarrow 2^{w/2 + 1} - 2 - m_1 - m_2 \]

verification: \[ g(y_i[0] \parallel y_i[1] \parallel y_i[2] \parallel z_i) = z_{i-1} \]
DKSS Signature Generation and Verification

\[ m = m_1 \parallel m_2 \]

\[ y_{i[0]} \]

\[ f^{2^{\frac{w}{2}} - 1 - m_1}(\alpha_1) \]

\[ f^{m_1}(x_{i[0]}) \]

\[ x_{i[0]} \]

\[ \alpha_1 \]

\[ \alpha_2 \]

\[ y_{i[1]} \]

\[ f^{2^{\frac{w}{2}} - 1 - m_2}(\alpha_2) \]

\[ f^{m_2}(x_{i[1]}) \]

\[ x_{i[1]} \]

\[ y_{i[2]} \]

\[ f^{2^{\frac{w}{2}} + 1 - 2 - c}(\alpha_3) \]

\[ f^{c}(x_{i[2]}) \]

\[ x_{i[2]} \]

\[ c \leftarrow 2^{\frac{w}{2} + 1} - 2 - m_1 - m_2 \]

verification: \( g(y_{i[0]} \parallel y_{i[1]} \parallel y_{i[2]} \parallel z_i) = z_{i-1} \)?
$m = m_1 \| m_2$

$y_i[0]$  $f^{2^{\frac{w}{2}} - 1 - m_1}(\alpha_1)$  $x_i[0]$  $f^{m_1}(x_i[0])$

$y_i[1]$  $\alpha_2$  $x_i[1]$  $f^{m_2}(x_i[1])$

$y_i[2]$  $f^{2^{\frac{w}{2}} - 1 - m_2}(\alpha_2)$  $x_i[2]$  $f^{2^{\frac{w}{2} + 1} - 2 - c}(\alpha_3)$

$c \leftarrow 2^{\frac{w}{2} + 1} - 2 - m_1 - m_2$

verification: $g(y_i[0] \| y_i[1] \| y_i[2] \| z_i) = z_{i-1}$
$$m = m_1 \| m_2$$

$$y_i[0]$$

$$x_i[0]$$

$$\alpha_1$$

$$f^{m_1}(x_i[0])$$

$$f^{2^{\frac{w}{2}}-1-m_1}(\alpha_1)$$

$$y_i[1]$$

$$x_i[1]$$

$$\alpha_2$$

$$f^{m_2}(x_i[1])$$

$$f^{2^{\frac{w}{2}}-1-m_2}(\alpha_2)$$

$$y_i[2]$$

$$x_i[2]$$

$$\alpha_3$$

$$f^{c}(x_i[2])$$

$$f^{2^{\frac{w}{2}}+1-2-c}(\alpha_3)$$

$$c \leftarrow 2^{\frac{w}{2}+1} - 2 - m_1 - m_2$$

verication: $$g(y_i[0] \| y_i[1] \| y_i[2] \| z_i) = z_{i-1}$$
m = m_1 || m_2

\[ y_i[0] \]
\[ f^{2\frac{w}{2} - 1 - m_1}(\alpha_1) \]
\[ \alpha_1 \]
\[ f^{m_1}(x_i[0]) \]
\[ x_i[0] \]

\[ y_i[1] \]
\[ f^{2\frac{w}{2}} - 1 - m_2(\alpha_2) \]
\[ \alpha_2 \]
\[ f^{m_2}(x_i[1]) \]
\[ x_i[1] \]

\[ y_i[2] \]
\[ f^{2\frac{w}{2} + 1 - 2 - c}(\alpha_3) \]
\[ \alpha_3 \]
\[ f^c(x_i[2]) \]
\[ x_i[2] \]

\[ c \leftarrow 2^{\frac{w}{2} + 1} - 2 - m_1 - m_2 \]

verification: \( g(y_i[0] || y_i[1] || y_i[2] || z_i) = z_{i - 1} \)
DKSS Signature Generation and Verification

\[ m = m_1\| m_2 \]

\( y_i[0] \)
\( f^{2^{\frac{w}{2}} - 1 - m_1}(\alpha_1) \)
\( f^{m_1}(x_i[0]) \)
\( x_i[0] \)
\( c \leftarrow 2^{\frac{w}{2}} + 1 - 2 - m_1 - m_2 \)

\( y_i[1] \)
\( f^{2^{\frac{w}{2}} - 1 - m_2}(\alpha_2) \)
\( f^{m_2}(x_i[1]) \)
\( x_i[1] \)

\( y_i[2] \)
\( f^{2^{\frac{w}{2}} + 1 - 2 - c}(\alpha_3) \)
\( f^{c}(x_i[2]) \)
\( x_i[2] \)

verification: \( g(y_i[0]\|y_i[1]\|y_i[2]\|z_i) = z_{i-1} \)?
Correction of the PRNG Specification

- in the original DKSS paper:
  - \( \text{PRNG}(\psi) = (\text{rand}, \psi') = (f(\psi), f(\psi) + \psi + 1 \mod 2^n) \)
  - forward secure
  - cannot be realized:

\[
\begin{align*}
Z_0 & \leftarrow Z_1 & \cdots & \leftarrow Z_{l-1} & \leftarrow Z_l \\
Y_1 & \leftarrow Y_2 & \cdots & \leftarrow Y_{l-1} & \leftarrow Y_l \\
X_1 & \leftarrow X_2 & \cdots & \leftarrow X_{l-1} & \leftarrow X_l
\end{align*}
\]

\[ \text{ thus: } \text{rand} \leftarrow \text{PRNG}(i, j) \]
Correction of the PRNG Specification

- in the original DKSS paper:
  - \( \text{PRNG}(\psi) = (\text{rand}, \psi') = (f(\psi), f(\psi) + \psi + 1 \mod 2^n) \)
  - forward secure
  - cannot be realized:

```
\[
\begin{array}{cccccc}
  Z_0 & \leftarrow & Z_1 & \leftarrow & Z_2 & \leftarrow & \ldots & \leftarrow & Z_{l-1} & \leftarrow & Z_l \\
  X_1 & \text{PRNG} & X_2 & \text{PRNG} & \ldots & \text{PRNG} & X_l \\
  Y_1 & \text{PRNG} & Y_2 & \text{PRNG} & \ldots & \text{PRNG} & Y_l \\
\end{array}
\]
```

- thus: \( \text{rand} \leftarrow \text{PRNG}(i, j) \)
Correction of the PRNG Specification

- in the original DKSS paper:
  - \( \text{PRNG}(\psi) = (\text{rand, } \psi') = (f(\psi), f(\psi) + \psi + 1 \mod 2^n) \)
- forward secure
- cannot be realized:

\[
\begin{align*}
  &\text{signature generation} \\
  &Z_0 \leftarrow Z_1 \leftarrow Z_2 \leftarrow \cdots \leftarrow Z_{l-1} \leftarrow Z_l \\
  &Y_1 \leftarrow Y_2 \leftarrow \cdots \leftarrow Y_l \\
  &X_1 \leftarrow X_2 \leftarrow \cdots \leftarrow X_l \\
\end{align*}
\]

\( \text{thus: } \text{rand} \leftarrow \text{PRNG}(i, j) \)
Correction of the PRNG Specification

- in the original DKSS paper:
  - \( \text{PRNG}(\psi) = (\text{rand}, \psi') = (f(\psi), f(\psi) + \psi + 1 \ mod \ 2^n) \)
- forward secure
- cannot be realized:

\[
\begin{align*}
\text{signature generation} & \\
Z_0 & \leftarrow Z_1 \leftarrow Z_2 \leftarrow \ldots \leftarrow Z_{l-1} \leftarrow Z_l \\
Y_1 & \leftarrow Y_2 \leftarrow \ldots \leftarrow Y_l \\
X_1 & \leftarrow X_2 \leftarrow \ldots \leftarrow X_l \\
\text{thus: } \text{rand} & \leftarrow \text{PRNG}(i, j)
\end{align*}
\]
in the original DKSS paper:

$$\text{PRNG}(\psi) = (\text{rand}, \psi') = (f(\psi), f(\psi) + \psi + 1 \mod 2^n)$$

forward secure
cannot be realized:

Thus: \(\text{rand} \leftarrow \text{PRNG}(i, j)\)
Correction of the PRNG Specification

- in the original DKSS paper:
  - \( \text{PRNG}(\psi) = (\text{rand}, \psi') = (f(\psi), f(\psi) + \psi + 1 \mod 2^n) \)
- forward secure
- cannot be realized:

\[
\begin{align*}
Z_0 & \xrightarrow{\text{PRNG}} Z_1 & \cdots & \xrightarrow{\text{PRNG}} Z_{l-1} & \xrightarrow{\text{PRNG}} Z_l \\
Y_1 & \xrightarrow{\text{PRNG}} Y_2 & \cdots & \xrightarrow{\text{PRNG}} Y_{l-1} & \xrightarrow{\text{PRNG}} Y_l \\
X_1 & \xrightarrow{\text{PRNG}} X_2 & \cdots & \xrightarrow{\text{PRNG}} X_{l-1} & \xrightarrow{\text{PRNG}} X_l
\end{align*}
\]

- thus: \( \text{rand} \leftarrow \text{PRNG}(i, j) \)
in the original DKSS paper:
PRNG(ψ) = (rand, ψ') = (f(ψ), f(ψ) + ψ + 1 mod 2^n)
forward secure
cannot be realized:

thus: rand ← PRNG(i, j)
Correction of the PRNG Specification

- in the original DKSS paper:
  \[ \text{PRNG}(\psi) = (\text{rand}, \psi') = (f(\psi), f(\psi) + \psi + 1 \mod 2^n) \]
- forward secure
- cannot be realized:

  ![Diagram of signature generation process]

  thus: \( \text{rand} \leftarrow \text{PRNG}(i, j) \)
in the original DKSS paper:
PRNG(ψ) = (rand, ψ′) = (f(ψ), f(ψ) + ψ + 1 mod 2^n)
forward secure
cannot be realized:

thus: rand ← PRNG(i, j)
Correction of the PRNG Specification

- in the original DKSS paper:
  \[ \text{PRNG}(\psi) = (\text{rand}, \psi') = (f(\psi), f(\psi) + \psi + 1 \mod 2^n) \]
- forward secure
- cannot be realized:

\[
\begin{align*}
\text{signature generation} \\
\bullet & \quad \cdots \quad \bullet \\
Z_0 & \quad Z_1 & \quad Z_2 & \quad \cdots & \quad Z_{l-1} & \quad Z_l \\
Y_1 & \quad Y_2 & \quad \cdots & \quad Z_{l-1} & \quad Z_l \\
X_1 & \quad \text{PRNG} & \quad X_2 & \quad \text{PRNG} & \quad \cdots & \quad \text{PRNG} & \quad X_l
\end{align*}
\]

- thus: \( \text{rand} \leftarrow \text{PRNG}(i, j) \)
Correction of the PRNG Specification

- in the original DKSS paper:
  \[ \text{PRNG}(\psi) = (\text{rand}, \psi') = (f(\psi), f(\psi) + \psi + 1 \mod 2^n) \]
  - forward secure
  - cannot be realized:

```
 signature generation

Z_0 ← Z_1 ← Z_2 ← \ldots ← Z_{l-1} ← Z_l
Y_1 ← Y_2 ← \ldots ← Y_l
X_1 \xrightarrow{\text{PRNG}} X_2 \xrightarrow{\text{PRNG}} \ldots \xrightarrow{\text{PRNG}} X_l
```

- thus: \( \text{rand} \leftarrow \text{PRNG}(i, j) \)
Correction of the PRNG Specification

- in the original DKSS paper:
  - \( \text{PRNG}(\psi) = (\text{rand}, \psi') = (f(\psi), f(\psi) + \psi + 1 \mod 2^n) \)
  - forward secure
  - cannot be realized:

```
signature generation
```

```
\begin{align*}
Z_0 & \leftarrow Z_1 \leftarrow Z_2 \leftarrow \cdots \leftarrow Z_{l-1} \leftarrow Z_l \\
Y_1 & \leftarrow Y_2 \leftarrow Y_3 \leftarrow \cdots \leftarrow Y_{l-1} \leftarrow Y_l \\
X_1 & \xrightarrow{\text{PRNG}} X_2 \xrightarrow{\text{PRNG}} \cdots \xrightarrow{\text{PRNG}} X_{l-1} \xleftarrow{\text{PRNG}} X_l
\end{align*}
```

- thus: \( \text{rand} \leftarrow \text{PRNG}(i,j) \)
Correction of the PRNG Specification

- in the original DKSS paper:
  \[ \text{PRNG}(\psi) = (\text{rand}, \psi') = (f(\psi), f(\psi) + \psi + 1 \mod 2^n) \]
- forward secure
- cannot be realized:

\[
\begin{align*}
\text{signature generation} \\
\bullet & \quad \cdots \quad \bullet \\
Z_0 & \quad Z_1 \quad Z_2 \quad \cdots \quad Z_{l-1} \quad \leftarrow \quad Z_l \\
\uparrow & \quad \uparrow \quad \text{PRNG} \quad \uparrow \quad \text{PRNG} \quad \cdots \quad \uparrow \quad \text{PRNG} \\
Y_1 & \quad Y_2 \quad \cdots \quad Y_l \\
\rightarrow & \quad \rightarrow \quad \rightarrow \quad \rightarrow \quad \rightarrow \\
X_1 & \quad X_2 \quad \cdots \quad X_l
\end{align*}
\]

- thus: \( \text{rand} \leftarrow \text{PRNG}(i, j) \)
Multiple Chains

\[ z_0 \leftarrow z_1 \leftarrow z_2 \leftarrow \cdots \leftarrow z_{r-1} \leftarrow z_r \]
\[ Y_1, Y_2 \]
\[ X_1, X_2 \]

\[ z'_0 \leftarrow z'_1 \leftarrow z'_2 \leftarrow \cdots \leftarrow z'_{r-1} \leftarrow z'_r \]
\[ Y'_0, Y'_1, Y'_2 \]
\[ X'_0, X'_1, X'_2 \]

- advantage: faster computation
- disadvantage: more memory
Multiple Chains

- advantage: faster computation
- disadvantage: more memory
1 Introduction

2 Preliminaries

3 DKSS

4 Implementation

5 Conclusion
Platform

- Tmote Sky WSN platform
- MSP430 16-bit Microcontroller
- 10 KB RAM
- 48 KB flash memory
- 8 MHz CPU speed
- OS: Contiki OS
Platform

- Tmote Sky WSN platform
- MSP430 16-bit Microcontroller
  - 10 KB RAM
  - 48 KB flash memory
  - 8 MHz CPU speed
- OS: Contiki OS
Platform

- Tmote Sky WSN platform
- MSP430 16-bit Microcontroller
- 10 KB RAM
- 48 KB flash memory
- 8 MHz CPU speed
- OS: Contiki OS
Platform

- Tmote Sky WSN platform
- MSP430 16-bit Microcontroller
- 10 KB RAM
- 48 KB flash memory
- 8 MHz CPU speed
- OS: Contiki OS
Platform

- Tmote Sky WSN platform
- MSP430 16-bit Microcontroller
- 10 KB RAM
- 48 KB flash memory
- 8 MHz CPU speed
- OS: Contiki OS
Platform

- Tmote Sky WSN platform
- MSP430 16-bit Microcontroller
- 10 KB RAM
- 48 KB flash memory
- 8 MHz CPU speed
- OS: Contiki OS
Block Cipher Choices

- **AES**
  - Fast
  - Block size = 128 bit (needed: 80 bit)
  - Needs tables (S-Box and MixColumn)
  - Unquestioned security

- **XXTEA**
  - Slower
  - Block size = 96 bit possible
  - No tables
  - Questionable security
Block Cipher Choices

- **AES**
  - fast
  - block size = 128 bit (needed: 80 bit)
  - needs tables (S-Box and MixColumn)
  - unquestioned security

- **XXTEA**
  - slower
  - block size = 96 bit possible
  - no tables
  - questionable security
Block Cipher Choices

- **AES**
  - fast
  - block size = 128 bit (needed: 80 bit)
  - needs tables (S-Box and MixColumn)
  - unquestioned security

- **XXTEA**
  - slower
  - block size = 96 bit possible
  - no tables
  - questionable security
Block Cipher Choices

- **AES**
  - fast
  - block size = 128 bit (needed: 80 bit)
  - needs tables (S-Box and MixColumn)
  - unquestioned security

- **XXTEA**
  - slower
  - block size = 96 bit possible
  - no tables
  - questionable security
Block Cipher Choices

- **AES**
  - fast
  - block size = 128 bit (needed: 80 bit)
  - needs tables (S-Box and MixColumn)
  - unquestioned security

- **XXTEA**
  - slower
  - block size = 96 bit possible
  - no tables
  - questionable security
Block Cipher Choices

- **AES**
  - fast
  - block size = 128 bit (needed: 80 bit)
  - needs tables (S-Box and MixColumn)
  - unquestioned security

- **XXTEA**
  - slower
  - block size = 96 bit possible
  - no tables
  - questionable security
Block Cipher Choices

- **AES**
  - fast
  - block size = 128 bit (needed: 80 bit)
  - needs tables (S-Box and MixColumn)
  - unquestioned security

- **XXTEA**
  - slower
  - block size = 96 bit possible
  - no tables
  - questionable security
Block Cipher Choices

- **AES**
  - fast
  - block size = 128 bit (needed: 80 bit)
  - needs tables (S-Box and MixColumn)
  - unquestioned security

- **XXTEA**
  - slower
  - block size = 96 bit possible
  - no tables
  - questionable security
Block Cipher Choices

- AES
  - fast
  - block size = 128 bit (needed: 80 bit)
  - needs tables (S-Box and MixColumn)
  - unquestioned security

- XXTEA
  - slower
  - block size = 96 bit possible
  - no tables
  - questionable security
Block Cipher Choices

- **AES**
  - fast
  - block size = 128 bit (needed: 80 bit)
  - needs tables (S-Box and MixColumn)
  - unquestioned security

- **XXTEA**
  - slower
  - block size = 96 bit possible
  - no tables
  - questionable security
Code Sizes

ROM size /10^3 byte

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>ROM Size (x10^3 byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>6.8</td>
</tr>
<tr>
<td>XXTEA</td>
<td>8.3</td>
</tr>
<tr>
<td>TinyECC</td>
<td>8.2</td>
</tr>
</tbody>
</table>
Multiple Chains

- time-memory tradeoff employing multiple chains
- \( l = 1024 \)
- AES

<table>
<thead>
<tr>
<th></th>
<th>1 chain</th>
<th>8 chains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign. Gen. time</td>
<td>5045 ms</td>
<td>4611 ms</td>
</tr>
<tr>
<td>Private Key</td>
<td>174 Bytes</td>
<td>924 Bytes</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
Z_0 &\leftarrow Z_1 \leftarrow Z_2 \leftarrow \cdots \leftarrow Z_{r-1} \leftarrow Z_r \\
Y_1 &\uparrow \quad Y_2 \\
X_1 &\uparrow \quad X_2
\end{align*}
\]

\[
\begin{align*}
Z'_0 &\leftarrow Z'_1 \leftarrow Z'_2 \leftarrow \cdots \leftarrow Z'_{r-1} \leftarrow Z'_r \\
Y'_0 &\uparrow \quad Y'_1 \quad Y'_2 \quad Y'_r \\
X'_0 &\uparrow \quad X'_1 \quad X'_2 \quad X'_r
\end{align*}
\]
Multiple Chains

- time-memory tradeoff employing multiple chains
- \( l = 1024 \)
- AES

<table>
<thead>
<tr>
<th></th>
<th>1 chain</th>
<th>8 chains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign. Gen. time</td>
<td>5045 ms</td>
<td>4611 ms</td>
</tr>
<tr>
<td>Private Key</td>
<td>174 Bytes</td>
<td>924 Bytes</td>
</tr>
</tbody>
</table>

\[
\begin{array}{cccc}
Z_0 & Z_1 & Z_2 & \cdots & Z_{r-1} & Z_r \\
Y_1 & Y_2 & \uparrow & \uparrow & \cdots & \uparrow \\
X_1 & X_2 & \leftarrow & \leftarrow & \cdots & \leftarrow \\
Y'_0 & Y'_1 & Y'_2 & \cdots & Y'_r \\
X'_0 & X'_1 & X'_2 & \cdots & X'_r \\
\end{array}
\]
Multiple Chains

- time-memory tradeoff employing multiple chains
- \( l = 1024 \)
- AES

<table>
<thead>
<tr>
<th></th>
<th>1 chain</th>
<th>8 chains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign. Gen. time</td>
<td>5045 ms</td>
<td>4611 ms</td>
</tr>
<tr>
<td>Private Key</td>
<td>174 Bytes</td>
<td>924 Bytes</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
Z_0 &\leftarrow Z_1 \leftarrow Z_2 \leftarrow \cdots \leftarrow Z_{r-1} \leftarrow Z_r \\
Y_1 &\uparrow \\
X_1 &
\end{align*}
\]

\[
\begin{align*}
Z'_0 &\leftarrow Z'_1 \leftarrow Z'_2 \leftarrow \cdots \leftarrow Z'_{r-1} \leftarrow Z'_r \\
Y'_n &\uparrow \\
X'_r &
\end{align*}
\]
<table>
<thead>
<tr>
<th>Message bit length $w$</th>
<th>8 bits</th>
<th>16 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XXTEA</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sign</td>
<td>0.383</td>
<td>5.770</td>
</tr>
<tr>
<td>Verify</td>
<td>0.098</td>
<td>1.394</td>
</tr>
<tr>
<td><strong>AES</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sign</td>
<td>0.279</td>
<td>4.135</td>
</tr>
<tr>
<td>Verify</td>
<td>0.071</td>
<td>1.007</td>
</tr>
</tbody>
</table>
Signature Generation Times

Legend
- ECDSA sign
- DKSS with AES 1_1024 sign

Mourier, Stampp, Strenzke: Implementation of Hash-Chain Signatures
Signature Verification Times

![Signature Verification Times Graph]

Legend
- ECDSA verif
- DKSS with XXTEA
- DKSS with AES

Table:

<table>
<thead>
<tr>
<th>Time (seconds)</th>
<th>Message Bit Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1.5</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2.5</td>
<td>0</td>
</tr>
</tbody>
</table>

Mourier, Stampp, Strenzke: Implementation of Hash-Chain Signatures
• compact implementation possible
• speed predictions from original paper not met
• narrow application conditions because of message size
• → integration with broadcast protocol difficult
• most efficient public key signature scheme for short messages
• speedup through AES coprocessor!
Conclusion

- compact implementation possible
- speed predictions from original paper not met
- narrow application conditions because of message size
  → integration with broadcast protocol difficult
- most efficient public key signature scheme for short messages
- speedup through AES coprocessor!

Mourier, Stampf, Strenzke: Implementation of Hash-Chain Signatures
Conclusion

- compact implementation possible
- speed predictions from original paper not met
- narrow application conditions because of message size
- integration with broadcast protocol difficult
- most efficient public key signature scheme for short messages
- speedup through AES coprocessor!
Conclusion

- compact implementation possible
- speed predictions from original paper not met
- narrow application conditions because of message size
- → integration with broadcast protocol difficult
- most efficient public key signature scheme for short messages
- speedup through AES coprocessor!
• compact implementation possible
• speed predictions from original paper not met
• narrow application conditions because of message size
  → integration with broadcast protocol difficult
• most efficient public key signature scheme for short messages
  • speedup through AES coprocessor!
compact implementation possible
speed predictions from original paper not met
narrow application conditions because of message size
→ integration with broadcast protocol difficult
most efficient public key signature scheme for short messages
speedup through AES coprocessor!
Thank You!