Towards Standardization of Threshold Schemes for Cryptographic Primitives at NIST

Luís Brandão
Joint work with: Apostol Vassilev, Nicky Mouha, Michael Davidson

National Institute of Standards and Technology (Gaithersburg MD, USA)

Presentation at ICMC19
International Cryptographic Module Conference
May 16, 2019 @ Vancouver, Canada
Outline

1. Introduction
2. Preliminaries
3. Step 1: NISTIR
4. Step 2: NTCW
5. Step 3: preliminary roadmap
6. Final remarks
Outline

1. Introduction

2. Preliminaries

3. Step 1: NISTIR

4. Step 2: NTCW

5. Step 3: preliminary roadmap

6. Final remarks
Should we share a secret?

Proverbial wisdom tells us to be careful.

- Three may keep a secret, if two of them are dead. (In: "Poor Richard's Almanack." Benjamin Franklin, 1735)
- Two may keep counsel, putting one away. (In: "Romeo and Juliet." William Shakespeare, 1597)
- For three may kepe counseil if twain be away! (In: The Ten Commandments of Love. Geoffrey Chaucer, 1340–1400)

This is specially relevant for secret keys in modern cryptography.

Cryptography relies on:

- secrecy, correctness, availability...
- implementations that use keys to operate an algorithm.

Accessed: July 2018
Should we share a secret?

“Three may keep a secret

(In: “Poor Richard's Almanack.” Benjamin Franklin, 1735) [Sau34]

Two may keep counsel

(In: “Romeo and Juliet.” William Shakespeare, 1597) [Sha97]

“For three may kepe counseil

(In: The Ten Commandments of Love. Geoffrey Chaucer, 1340–1400) [Cha00]
Should we share a secret?

“Three may keep a secret, if two of them are dead.”
(In: “Poor Richard's Almanack.” Benjamin Franklin, 1735) [Sau34]

“Two may keep counsel, putting one away.”
(In: “Romeo and Juliet.” William Shakespeare, 1597) [Sha97]

“For three may kepe counseil if twain be away!”
(In: The Ten Commandments of Love. Geoffrey Chaucer, 1340–1400) [Cha00]
Should we share a secret?

Proverbial wisdom tells us to be careful

“Three may keep a secret, if two of them are dead.”
(In: “Poor Richard’s Almanack.” Benjamin Franklin, 1735) [Sau34]

“Two may keep counsel, putting one away.”
(In: “Romeo and Juliet.” William Shakespeare, 1597) [Sha97]

“For three may kepe counseil if twain be away!”
(In: The Ten Commandments of Love. Geoffrey Chaucer, 1340–1400) [Cha00]
Should we share a secret?

Proverbial wisdom tells us to be careful

“Three may keep a secret, if two of them are dead.”
(In: “Poor Richard's Almanack.” Benjamin Franklin, 1735) [Sau04]

“Two may keep counsel, putting one away.”
(In: “Romeo and Juliet.” William Shakespeare, 1597) [Sha97]

“For three may kepe counseil if twain be away!”
(In: The Ten Commandments of Love. Geoffrey Chaucer, 1340–1400) [Cha00]

This is specially relevant for secret keys in modern cryptography.
Should we share a secret?

Proverbial wisdom tells us to be careful

“Three may keep a secret, if two of them are dead.”
(In: “Poor Richard's Almanack.” Benjamin Franklin, 1735) [Sau04]

“Two may keep counsel, putting one away.”
(In: “Romeo and Juliet.” William Shakespeare, 1597) [Sha97]

“For three may kepe counsel if twain be away!”
(In: The Ten Commandments of Love. Geoffrey Chaucer, 1340–1400) [Cha00]

This is specially relevant for secret keys in modern cryptography.

Cryptography relies on:

➤ secrecy, correctness, availability ... of cryptographic keys
1. Introduction

Should we share a secret?

Proverbial wisdom tells us to be careful

“Three may keep a secret, if two of them are dead.”
(In: “Poor Richard’s Almanack.” Benjamin Franklin, 1735) [Sau34]

“Two may keep counsel, putting one away.”
(In: “Romeo and Juliet.” William Shakespeare, 1597) [Sha97]

“For three may kepe counseil if twain be away!”
(In: The Ten Commandments of Love. Geoffrey Chaucer, 1340–1400) [Cha00]

This is specially relevant for secret keys in modern cryptography.

Cryptography relies on:

- secrecy, correctness, availability ... of cryptographic keys
- implementations that use keys to operate an algorithm
1. Introduction

Should we share a secret?

Proverbial wisdom tells us to be careful

“Three may keep a secret, if two of them are dead.”
(In: “Poor Richard’s Almanack.” Benjamin Franklin, 1735) [Sau34]

“Two may keep counsel, putting one away.”
(In: “Romeo and Juliet.” William Shakespeare, 1597) [Sha97]

“For three may kepe counsel if twain be away!”
(In: The Ten Commandments of Love. Geoffrey Chaucer, 1340–1400) [Cha00]

This is specially relevant for secret keys in modern cryptography.

Cryptography relies on:

- secrecy, correctness, availability ... of cryptographic keys
- implementations that use keys to operate an algorithm
1. Introduction

Crypto can be affected by vulnerabilities!

Attacks can exploit differences between ideal vs. real implementations

- "Bellcore attack" (1997) [BDL97] [SH07]
- Cold-boot attacks (2009) [HSH+09] [Don13]
- Heartbleed bug (2014) [DLK+14] heartbleed.com
- "ZigBee Chain reaction" (2017) [RSWO17]
- Meltdown & Spectre (2017) [LSG+18, KGG+18] meltdownattack.com
- Foreshadow (2018) [BMW+18, WBM+18] foreshadowattack.eu

Also, operators of cryptographic implementations can go rogue.

How can we address single-points of failure?
Crypto can be affected by vulnerabilities!

Attacks can exploit differences between ideal vs. real implementations.
Crypto can be affected by vulnerabilities!

Attacks can exploit differences between ideal vs. real implementations


[BDL97]  [HSH+09]  [DLK+14]  [RSWO17]  [LSG+18, KGG+18]  [BMW+18, WBM+18]  [MDS19]
1. Introduction

Crypto can be affected by vulnerabilities!

 Attacks can exploit differences between ideal vs. real **implementations**

<table>
<thead>
<tr>
<th>Attack Name</th>
<th>Year</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Bellcore attack” (1997)</td>
<td>L97</td>
<td>BDL97</td>
</tr>
<tr>
<td>Cold-boot attacks (2009)</td>
<td>HSH+09</td>
<td></td>
</tr>
<tr>
<td>Heartbleed bug (2014)</td>
<td>DLK+14</td>
<td></td>
</tr>
<tr>
<td>“ZigBee Chain reaction” (2017)</td>
<td>RSWO17</td>
<td></td>
</tr>
<tr>
<td>Meltdown &amp; Spectre (2017)</td>
<td>LSG+18, KGG+18</td>
<td></td>
</tr>
<tr>
<td>Foreshadow (2018)</td>
<td>BMW+18, WBM+18</td>
<td></td>
</tr>
<tr>
<td>Microarchitectural Data Sampling (2019)</td>
<td>MDS19</td>
<td></td>
</tr>
</tbody>
</table>

Also, **operators** of cryptographic implementations can go rogue
Crypto can be affected by vulnerabilities!

Attacks can exploit differences between ideal vs. real implementations

“Bellcore attack” (1997) [BDL97]
Cold-boot attacks (2009) [HSH+09]
Heartbleed bug (2014) [DLK+14]
“ZigBee Chain reaction” (2017) [RSWO17]
Meltdown & Spectre (2017) [LSG+18, KGG+18]
Foreshadow (2018) [BMW+18, WBM+18]
Microarchitectural Data Sampling (2019) [MDS19]

Also, operators of cryptographic implementations can go rogue

How can we address single-points of failure?

*question-2.html
*4296.html
*colored-elephant.html

* = clker.com/clipart-
The threshold approach

At high-level: use redundancy & diversity to mitigate the compromise of up to a threshold number ($f$-out-of-$n$) of components.

The intuitive aim: improve security vs. a non-threshold scheme.

NIST - CSD wants to standardize threshold schemes for cryptographic primitives.

Potential primitives: signing, decryption, enciphering, key-generation, ...

Some properties:

- withstands several compromised components;
- needs several uncompromised components;
- prevents secret keys from being in one place;
- enhances resistance against side-channel attacks; ...
The threshold approach

At high-level:
use redundancy & diversity to mitigate the *compromise* of up to a threshold number *(f*-out-of-*) of components
The threshold approach

At high-level:
use redundancy & diversity to mitigate the compromise of up to a threshold number \((f\text{-out-of-}n)\) of components

The intuitive aim:
improve security vs. a non-threshold scheme
The threshold approach

At high-level:

use redundancy & diversity to mitigate the *compromise* of up to a threshold number \((f\text{-out-of-}n)\) of components

The intuitive aim:

improve security vs.

a non-threshold scheme

NIST-CSD wants to standardize *threshold schemes for cryptographic primitives*
The threshold approach

At high-level:
use redundancy & diversity to mitigate the compromise of up to a threshold number \( (f\text{-out-of-}n) \) of components

The intuitive aim:
improve security vs. a non-threshold scheme

NIST-CSD wants to standardize threshold schemes for cryptographic primitives

Potential primitives: signing, decryption, enciphering, key-generation, ...
The threshold approach

At high-level:
use redundancy & diversity to mitigate the compromise of up to a threshold number \((f\text{-out-of-}n)\) of components.

The intuitive aim:
improve security vs.
a non-threshold scheme.

NIST-CSD wants to standardize threshold schemes for cryptographic primitives.

Potential primitives: signing, decryption, enciphering, key-generation, ...

Some properties:

- **withstands** several *compromised* components;
- **needs** several *uncompromised* components;
- **prevents** secret keys from being in one place;
- **enhances** resistance against side-channel attacks; ...
Secret Sharing Schemes (a starting point)

Split a secret key into $n$ secret “shares” for storage at rest.

Shamir scheme (1979)\cite{Sha79}

\[
\lambda(x) = y = \Lambda(1), \Lambda(2), \Lambda(3)
\]

Alice

Bob

Cai

Example 2-out-of-n secret sharing

▶ The secret $y$ is placed in the $y$-axis;
▶ A random line $\Lambda$ is drawn crossing the secret;
▶ Each share is a point $(\Lambda(i), i)$ in the line $\Lambda$;
▶ Each share alone has no information about the secret.

Any pair of shares allows recovering the secret

But how to avoid recombining the key when the key is needed by an algorithm?

Use threshold schemes for cryptographic primitives (next)
Secret Sharing Schemes (a starting point)

Split a secret key into \( n \) secret “shares” for storage at rest.

Example 2-out-of-\( n \) secret sharing

- The secret \( y_s \) is placed in the \( y \)-axis;
Secret Sharing Schemes (a starting point)

Split a secret key into $n$ secret “shares” for storage at rest.

Example 2-out-of-$n$ secret sharing
- The secret $y_s$ is placed in the $y$-axis;
- A random line $\Lambda$ is drawn crossing the secret;
Secret Sharing Schemes (a starting point)

Split a secret key into \( n \) secret “shares” for storage at rest.

**Example 2-out-of-\( n \) secret sharing**
- The secret \( y_s \) is placed in the \( y \)-axis;
- A random line \( \Lambda \) is drawn crossing the secret;
- Each share is a point \((\Lambda(i), i)\) in the line \( \Lambda \);
Secret Sharing Schemes (a starting point)

Split a secret key into \( n \) secret “shares” for storage at rest.

**Example 2-out-of-\( n \) secret sharing**
- The secret \( y_s \) is placed in the \( y \)-axis;
- A random line \( \Lambda \) is drawn crossing the secret;
- Each share is a point \( (\Lambda(i), i) \) in the line \( \Lambda \);

Each share alone has no information about the secret.
Secret Sharing Schemes (a starting point)

Split a secret key into $n$ secret “shares” for storage at rest.

Example 2-out-of-$n$ secret sharing

- The secret $y_s$ is placed in the $y$-axis;
- A random line $\Lambda$ is drawn crossing the secret;
- Each share is a point $(\Lambda(i), i)$ in the line $\Lambda$;

Each share alone has no information about the secret.
Any pair of shares allows recovering the secret.
Secret Sharing Schemes (a starting point)

Split a secret key into $n$ secret “shares” for storage at rest.

Example 2-out-of-$n$ secret sharing
- The secret $y_s$ is placed in the $y$-axis;
- A random line $\Lambda$ is drawn crossing the secret;
- Each share is a point $(\Lambda(i), i)$ in the line $\Lambda$;

Each share alone has no information about the secret. Any pair of shares allows recovering the secret

But how to avoid recombinining the key when the key is needed by an algorithm?
Secret Sharing Schemes (a starting point)

Split a secret key into \( n \) secret “shares” for storage at rest.

Example 2-out-of-\( n \) secret sharing

- The secret \( y_s \) is placed in the \( y \)-axis;
- A random line \( \Lambda \) is drawn crossing the secret;
- Each share is a point \((\Lambda(i), i)\) in the line \( \Lambda \);

Each share alone has no information about the secret.
Any pair of shares allows recovering the secret

But how to avoid recombing the key when the key is needed by an algorithm?

Use threshold schemes for cryptographic primitives (next)
Goal(s) for this presentation

Overview the NIST effort towards standardization of threshold schemes
Goal(s) for this presentation

Overview the NIST effort towards standardization of threshold schemes

1. Convey high-dimensionality of the threshold space
Goal(s) for this presentation

Overview the NIST effort towards standardization of threshold schemes

1. Convey high-dimensionality of the threshold space
2. Describe the steps so far and ahead
Goal(s) for this presentation

Overview the NIST effort towards standardization of threshold schemes

1. Convey high-dimensionality of the threshold space

2. Describe the steps so far and ahead

3. Motivate feedback and engagement from stakeholders
Outline

1. Introduction
2. Preliminaries
3. Step 1: NISTIR
4. Step 2: NTCW
5. Step 3: preliminary roadmap
6. Final remarks
A simple example: RSA signature (or decryption) [RSA78]
A simple example: RSA signature (or decryption) [RSA78]

<table>
<thead>
<tr>
<th>Conventional scheme (k = n = 1)</th>
<th>A 3-out-of-3 threshold scheme (k = n = 3)</th>
</tr>
</thead>
</table>

Conventional scheme:

- KeyGen (by signer):
  - Public Modulus: \(N = p \cdot q\)
  - Secret SignKey: \(d\)
  - Public VerKey: \(e = d^{-1} \mod \phi\)

- Sign \((m)\):
  - \(\sigma = m^d \mod N\)

- Verify \((\sigma, m)\):
  - \(\sigma^e = ? m \mod N\)

A 3-out-of-3 threshold scheme:

- KeyGen (by dealer):
  - Same \(N, d, e\)
  - SubKeys: \(d_1, d_2, d_3\)
    - \(d_1 + d_2 + d_3 = d \mod \phi\)

- Sign \((m)\):
  - Separate:
    - \(s_i = m^{d_i} \mod N\) for \(i = 1, 2, 3\)
  - Combine:
    - \(\sigma = s_1 \cdot s_2 \cdot s_3 \mod N\)

- Verify \((\sigma, m)\):
  - \(\sigma^e = ? m \mod N\)

About this threshold scheme:

- SignKey \(d\) not recombined; can reshare leaving \(e\) fixed; same \(\sigma\); efficient!

Facilitating setting:

- \(\exists\) dealer;
- \(\exists\) homomorphism;
- all parties learn \(m\).

Not fault-tolerant: a single sub-signer can boycott a correct signing.

Can other threshold schemes be implemented:

- \(\not\exists\) dealer, \(\not\exists\) homomorphisms, secret-shared \(m\), withstanding \(f\) malicious signers?
  - Yes, using threshold cryptography (with more complicated schemes)
A simple example: RSA signature (or decryption) \[\text{[RSA78]}\]

<table>
<thead>
<tr>
<th>Conventional scheme ((k = n = 1))</th>
<th>A 3-out-of-3 threshold scheme ((k = n = 3))</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyGen</td>
<td>KeyGen</td>
</tr>
<tr>
<td>Sign</td>
<td>Sign</td>
</tr>
<tr>
<td>Verify</td>
<td>Verify</td>
</tr>
</tbody>
</table>

About this threshold scheme:
- Sign Key \(d\) not recombined; can reshare \(d\) leaving \(e\) fixed; same \(\sigma\); efficient!

Facilitating setting:
- \(\exists\) dealer;
- \(\exists\) homomorphism;
- all parties learn \(m\).

Not fault-tolerant: a single sub-signer can boycott a correct signing.

Can other threshold schemes be implemented:
- \(\nexists\) dealer,
- \(\nexists\) homomorphisms, secret-shared \(m\), withstanding \(f\) malicious signers?
- Yes, using threshold cryptography (with more complicated schemes).
## A simple example: RSA signature (or decryption) [RSA78]

### Conventional scheme \( (k = n = 1) \)
- **KeyGen**
  - Public Modulus: \( N = p \cdot q \)
  - Secret SignKey: \( d \)
  - Public VerKey: \( e \ (= d^{-1} \pmod{\phi}) \)
- **Sign** \((m)\): \( \sigma = m^d \pmod{N} \)
- **Verify** \((\sigma, m)\): \( \sigma^e = ? m \pmod{N} \)

### A 3-out-of-3 threshold scheme \( (k = n = 3) \)
- **KeyGen**
- **Sign**
- **Verify**
A simple example: RSA signature (or decryption) \cite{RSA78}

Conventional scheme \((k = n = 1)\)

- **KeyGen (by **signer**):**
  - Public Modulus: \(N = p \cdot q\)
  - Secret **SignKey**: \(d\)
  - Public VerKey: \(e \equiv d^{-1} \pmod{\phi}\)

- **Sign** (\(m\)): \(\sigma = m^d \pmod{N}\)

- **Verify** (\(\sigma, m\)): \(\sigma^e = m \pmod{N}\)

A 3-out-of-3 threshold scheme \((k = n = 3)\)

- **KeyGen (by **dealer**):**
  - Same \(N, d, e\)
  - SubKeys: \(d_1, d_2, d_3\) : \(d_1 + d_2 + d_3 = d \pmod{\phi}\)

- **Sign**

- **Verify**
## 2. Preliminaries

### A simple example: RSA signature (or decryption) [RSA78]

**Conventional scheme** \((k = n = 1)\)

- **KeyGen (by signer):**
  - Public Modulus: \(N = p \cdot q\)
  - Secret SignKey: \(d\)
  - Public VerKey: \(e = d^{-1} \pmod{\phi}\)
- **Sign**\((m)\): \(\sigma = m^d \pmod{N}\)
- **Verify**\((\sigma, m)\): \(\sigma^e \equiv m \pmod{N}\)

**A 3-out-of-3 threshold scheme** \((k = n = 3)\)

- **KeyGen (by dealer):**
  - Same \(N, d, e\)
  - SubKeys: \(d_1, d_2, d_3 : d_1 + d_2 + d_3 = d \pmod{\phi}\)
  - Sign\((m)\): 
    - separate: \(s_i = m^{d_i} \pmod{N} : i = 1, 2, 3\)
    - combine: \(\sigma = s_1 \cdot s_2 \cdot s_3 \pmod{N}\)
- **Verify**
A simple example: RSA signature (or decryption) [RSA78]

Conventional scheme \((k = n = 1)\)

- KeyGen (by signer):
  - Public Modulus: \(N = p \cdot q\)
  - Secret SignKey: \(d\)
  - Public VerKey: \(e \equiv d^{-1} \pmod{\phi}\)
- Sign \((m)\): \(\sigma = m^d \pmod{N}\)
- Verify \((\sigma, m)\): \(\sigma^e =? m \pmod{N}\)

A 3-out-of-3 threshold scheme \((k = n = 3)\)

- KeyGen (by dealer):
  - Same \(N, d, e\)
  - SubKeys: \(d_1, d_2, d_3\) : \(d_1 + d_2 + d_3 = d \pmod{\phi}\)
- Sign \((m)\): \(\{\) separate: \(s_i = m^{d_i} \pmod{N} : i = 1, 2, 3\)
  combine: \(\sigma = s_1 \cdot s_2 \cdot s_3 \pmod{N}\) \(\}\)
- Verify \((\sigma, m)\): \(\sigma^e =? m \pmod{N}\)

About this threshold scheme:
- SignKey \(d\) not recombined; can reshare leaving \(e\) fixed; same \(\sigma\); efficient!
- Facilitating setting: \(\exists\) dealer; \(\exists\) homomorphism; all parties learn \(m\).
- Not fault-tolerant: a single sub-signer can boycott a correct signing.

Can other threshold schemes be implemented: \(\nexists\) dealer, \(\nexists\) homomorphisms, secret-shared \(m\), withstanding \(f\) malicious signers?
Yes, using threshold cryptography (with more complicated schemes).
A simple example: RSA signature (or decryption) [RSA78]

Conventional scheme \((k = n = 1)\)

- **KeyGen** (by signer):
  - Public Modulus: \(N = p \cdot q\)
  - Secret SignKey: \(d\)
  - Public VerKey: \(e = d^{-1} \pmod{\phi}\)
- **Sign**(\(m\)): \(\sigma = m^d \pmod{N}\)
- **Verify**(\(\sigma, m\)): \(\sigma^e = \,? \, m \pmod{N}\)

A 3-out-of-3 threshold scheme \((k = n = 3)\)

- **KeyGen** (by dealer):
  - Same \(N, d, e\)
  - SubKeys: \(d_1, d_2, d_3 : d_1 + d_2 + d_3 = d \pmod{\phi}\)
- **Sign**(\(m\)): \(\{\text{separate}: s_i = m^{d_i} \pmod{N} : i = 1, 2, 3\}
  \text{combine}: \sigma = s_1 \cdot s_2 \cdot s_3 \pmod{N}\}
- **Verify**(\(\sigma, m\)): \(\sigma^e = \,? \, m \pmod{N}\)

About this threshold scheme:

SignKey \(d\) not recombined;

Conventional scheme (\(k = n = 1\))

- **KeyGen** (by signer):
  - Public Modulus: \(N = p \cdot q\)
  - Secret SignKey: \(d\)
  - Public VerKey: \(e = d^{-1} \pmod{\phi}\)
- **Sign**(\(m\)): \(\sigma = m^d \pmod{N}\)
- **Verify**(\(\sigma, m\)): \(\sigma^e = \,? \, m \pmod{N}\)

A 3-out-of-3 threshold scheme (\(k = n = 3\))

- **KeyGen** (by dealer):
  - Same \(N, d, e\)
  - SubKeys: \(d_1, d_2, d_3 : d_1 + d_2 + d_3 = d \pmod{\phi}\)
- **Sign**(\(m\)): \(\{\text{separate}: s_i = m^{d_i} \pmod{N} : i = 1, 2, 3\}
  \text{combine}: \sigma = s_1 \cdot s_2 \cdot s_3 \pmod{N}\}
- **Verify**(\(\sigma, m\)): \(\sigma^e = \,? \, m \pmod{N}\)

About this threshold scheme:

SignKey \(d\) not recombined;
A simple example: RSA signature (or decryption) [RSA78]

**Conventional scheme** \(k = n = 1\)
- **KeyGen (by signer):**
  - Public Modulus: \(N = p \cdot q\)
  - Secret SignKey: \(d\)
  - Public VerKey: \(e = d^{-1} \pmod{\phi}\)
- **Sign\((m)\):** \(\sigma = m^d \pmod{N}\)
- **Verify\((\sigma, m)\):** \(\sigma^e = \equiv m \pmod{N}\)

**A 3-out-of-3 threshold scheme** \(k = n = 3\)
- **KeyGen (by dealer):**
  - Same \(N, d, e\)
  - SubKeys: \(d_1, d_2, d_3 : d_1 + d_2 + d_3 = d \pmod{\phi}\)
- **Sign\((m)\):**
  - separate: \(s_i = m^{d_i} \pmod{N} : i = 1, 2, 3\)
  - combine: \(\sigma = s_1 \cdot s_2 \cdot s_3 \pmod{N}\)
- **Verify\((\sigma, m)\):** \(\sigma^e = \equiv m \pmod{N}\)

**About this threshold scheme:**
SignKey \(d\) not recombined; can **reshare** \(d\) leaving \(e\) fixed;
A simple example: RSA signature (or decryption) \cite{RSA78}

<table>
<thead>
<tr>
<th>Conventional scheme ((k = n = 1))</th>
<th>A 3-out-of-3 threshold scheme ((k = n = 3))</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyGen (by signer):</td>
<td>KeyGen (by dealer):</td>
</tr>
<tr>
<td></td>
<td>Same (N, d, e)</td>
</tr>
<tr>
<td>Public Modulus: (N = p \cdot q)</td>
<td>SubKeys: (d_1, d_2, d_3 : d_1 + d_2 + d_3 = d \pmod{\phi})</td>
</tr>
</tbody>
</table>
| Secret SignKey: \(d\)              | Sign\((m)\): \[
| Public VerKey: \(e = d^{-1} \pmod{\phi}\) | \{ \text{separate: } s_i = m^{d_i} \pmod{N} : i = 1, 2, 3 \}
| Sign\((m)\): \(\sigma = m^d \pmod{N}\) | combine: \(\sigma = s_1 \cdot s_2 \cdot s_3 \pmod{N}\) \} |
| Verify\((\sigma, m)\): \(\sigma^e =? m \pmod{N}\) | Verify\((\sigma, m)\): \(\sigma^e =? m \pmod{N}\) |

About this threshold scheme:

SignKey \(d\) not recombined; can \textit{reshare} \(d\) leaving \(e\) fixed; \textit{same} \(\sigma\);
A simple example: RSA signature (or decryption) [RSA78]

Conventional scheme \((k = n = 1)\)

- **KeyGen (by signer):**
  - Public Modulus: \(N = p \cdot q\)
  - Secret SignKey: \(d\)
  - Public VerKey: \(e \equiv d^{-1} \pmod{\phi}\)
- **Sign** \((m)\): \(\sigma = m^d \pmod{N}\)
- **Verify** \((\sigma, m)\): \(\sigma^e \overset{?}{=} m \pmod{N}\)

A 3-out-of-3 threshold scheme \((k = n = 3)\)

- **KeyGen (by dealer):**
  - Same \(N, d, e\)
  - SubKeys: \(d_1, d_2, d_3 : d_1 + d_2 + d_3 = d \pmod{\phi}\)
- **Sign** \((m)\): \(\{\text{separate: } s_i = m^{d_i} \pmod{N} : i = 1, 2, 3\}
  - combine: \(\sigma = s_1 \cdot s_2 \cdot s_3 \pmod{N}\) \}
- **Verify** \((\sigma, m)\): \(\sigma^e \overset{?}{=} m \pmod{N}\)

About this threshold scheme:

SignKey \(d\) not recombined; can reshare \(d\) leaving \(e\) fixed; same \(\sigma\); efficient!
A simple example: RSA signature (or decryption) [RSA78]

<table>
<thead>
<tr>
<th>Conventional scheme ( (k = n = 1) )</th>
<th>A 3-out-of-3 threshold scheme ( (k = n = 3) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyGen (by signer):</td>
<td>KeyGen (by dealer):</td>
</tr>
<tr>
<td>Public Modulus: ( N = p \cdot q )</td>
<td>Same ( N, d, e )</td>
</tr>
<tr>
<td>Secret SignKey: ( d )</td>
<td>SubKeys: ( d_1, d_2, d_3 : d_1 + d_2 + d_3 = d \mod \phi )</td>
</tr>
</tbody>
</table>
| Public VerKey: \( e = d^{-1} \mod \phi \) | Sign \( (m) \): \begin{cases} separate: & s_i = m^{d_i} \mod N \mod : i = 1, 2, 3 \\ combine: & \sigma = s_1 \cdot s_2 \cdot s_3 \mod N \end{cases} \}
| Sign \( (m) \): \( \sigma = m^d \mod N \) | Verify \( (\sigma, m) \): \( \sigma^e = m \mod N \) |
| Verify \( (\sigma, m) \): \( \sigma^e = m \mod N \) | Verify \( (\sigma, m) \): \( \sigma^e = m \mod N \) |

About this threshold scheme:

SignKey \( d \) not recombined; can **reshare** \( d \) leaving \( e \) fixed; same \( \sigma \); efficient!

Facilitating setting: \( \exists \) dealer;
### A simple example: RSA signature (or decryption) [RSA78]

<table>
<thead>
<tr>
<th><strong>Conventional scheme</strong> (k = n = 1)</th>
<th><strong>A 3-out-of-3 threshold scheme</strong> (k = n = 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyGen (by signer):</td>
<td>KeyGen (by dealer):</td>
</tr>
<tr>
<td>Public Modulus: (N = p \cdot q)</td>
<td>Same (N, d, e)</td>
</tr>
<tr>
<td>Secret SignKey: (d)</td>
<td>SubKeys: (d_1, d_2, d_3 : d_1 + d_2 + d_3 = d \pmod{\phi})</td>
</tr>
</tbody>
</table>
| Public VerKey: \(e = d^{-1} \pmod{\phi}\) | Sign \((m)\): \{ separate: \(s_i = m^{d_i} \pmod{N} : i = 1, 2, 3\)  
| Sign \((m)\): \(\sigma = m^d \pmod{N}\)  | combine: \(\sigma = s_1 \cdot s_2 \cdot s_3 \pmod{N}\} \} |
| Verify \((\sigma, m)\): \(\sigma^e = ? m \pmod{N}\) | Verify \((\sigma, m)\): \(\sigma^e = ? m \pmod{N}\) |

**About this threshold scheme:**

SignKey \(d\) not recombined; can *reshare* \(d\) leaving \(e\) fixed; same \(\sigma\); efficient!

**Facilitating setting:** \(\exists\) dealer; \(\exists\) homomorphism;
## A simple example: RSA signature (or decryption) [RSA78]

### Conventional scheme \((k = n = 1)\)
- **KeyGen** (by signer):
  - Public Modulus: \(N = p \cdot q\)
  - Secret SignKey: \(d\)
  - Public VerKey: \(e = d^{-1} \pmod{\phi}\)
- **Sign** \((m)\):
  \[\sigma = m^d \pmod{N}\]
- **Verify** \((\sigma, m)\):
  \[\sigma^e = ? m \pmod{N}\]

### A 3-out-of-3 threshold scheme \((k = n = 3)\)
- **KeyGen** (by dealer):
  - Same \(N, d, e\)
  - SubKeys: \(d_1, d_2, d_3 : d_1 + d_2 + d_3 = d \pmod{\phi}\)
- **Sign** \((m)\): 
  - separate: \(s_i = m^{d_i} \pmod{N} : i = 1, 2, 3\)
  - combine: \(\sigma = s_1 \cdot s_2 \cdot s_3 \pmod{N}\)
- **Verify** \((\sigma, m)\):
  \[\sigma^e = ? m \pmod{N}\]

### About this threshold scheme:
- SignKey \(d\) not recombined; can *reshare* \(d\) leaving \(e\) fixed; same \(\sigma\); efficient!

**Facilitating setting:** \(\exists\) dealer; \(\exists\) homomorphism; **all parties learn** \(m\).
2. Preliminaries

A simple example: RSA signature (or decryption) [RSA78]

Conventional scheme \((k = n = 1)\)

- **KeyGen** (by signer):
  - Public Modulus: \(N = p \cdot q\)
  - Secret SignKey: \(d\)
  - Public VerKey: \(e \equiv d^{-1} \pmod{\phi}\)

- **Sign** \((m)\): \(\sigma = m^d \pmod{N}\)

- **Verify** \((\sigma, m)\): \(\sigma^e \equiv m \pmod{N}\)

A 3-out-of-3 threshold scheme \((k = n = 3)\)

- **KeyGen** (by dealer):
  - Same \(N, d, e\)
  - SubKeys: \(d_1, d_2, d_3 : d_1 + d_2 + d_3 = d \pmod{\phi}\)

- **Sign** \((m)\): \(\{\text{separate}: s_i = m^{d_i} \pmod{N} : i = 1, 2, 3\}
  \text{combine: } \sigma = s_1 \cdot s_2 \cdot s_3 \pmod{N}\}

- **Verify** \((\sigma, m)\): \(\sigma^e \equiv m \pmod{N}\)

About this threshold scheme:

SignKey \(d\) not recombined; can *reshare* \(d\) leaving \(e\) fixed; same \(\sigma\); efficient!

**Facilitating setting:** \exists dealer; \exists homomorphism; all parties learn \(m\).

**Not fault-tolerant:** a single sub-signer can boycott a correct signing.
A simple example: RSA signature (or decryption) [RSA78]

Conventional scheme \((k = n = 1)\)

- **KeyGen (by signer):**
  - Public Modulus: \(N = p \cdot q\)
  - Secret SignKey: \(d\)
  - Public VerKey: \(e \equiv d^{-1} (\mod \phi)\)

- **Sign\((m)\):** \(\sigma = m^d \pmod{N}\)

- **Verify\((\sigma, m)\):** \(\sigma^e \equiv m \pmod{N}\)

A 3-out-of-3 threshold scheme \((k = n = 3)\)

- **KeyGen (by dealer):**
  - Same \(N, d, e\)
  - SubKeys: \(d_1, d_2, d_3 : d_1 + d_2 + d_3 = d \pmod{\phi}\)

- **Sign\((m)\):** \(\sigma = s_1 \cdot s_2 \cdot s_3 \pmod{N}\)

- **Verify\((\sigma, m)\):** \(\sigma^e \equiv m \pmod{N}\)

About this threshold scheme:

SignKey \(d\) not recombined; can *reshare* \(d\) leaving \(e\) fixed; same \(\sigma\); efficient!

Facilitating setting: \(\exists\) dealer; \(\exists\) homomorphism; all parties learn \(m\).

Not fault-tolerant: a single sub-signer can boycott a correct signing.

Can other threshold schemes be implemented?
A simple example: RSA signature (or decryption) [RSA78]

<table>
<thead>
<tr>
<th>Conventional scheme ( (k = n = 1) )</th>
<th>A 3-out-of-3 threshold scheme ( (k = n = 3) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyGen (by signer):</td>
<td>KeyGen (by dealer):</td>
</tr>
<tr>
<td>▶ Public Modulus: ( N = p \cdot q )</td>
<td>▶ Same ( N, d, e )</td>
</tr>
<tr>
<td>▶ Secret SignKey: ( d )</td>
<td>▶ SubKeys: ( d_1, d_2, d_3 ) : ( d_1 + d_2 + d_3 = d \mod \phi )</td>
</tr>
<tr>
<td>▶ Public VerKey: ( e ) ( = d^{-1} \mod \phi )</td>
<td>▶ Sign( (m) ): { separate: ( s_i = m^{d_i} \mod N ) : ( i = 1, 2, 3 ) } combine: ( \sigma = s_1 \cdot s_2 \cdot s_3 \mod N ) }</td>
</tr>
<tr>
<td>▶ Sign( (m) ): ( \sigma = m^d \mod N )</td>
<td>▶ Verify( (\sigma, m) ): ( \sigma^e = ? m \mod N )</td>
</tr>
<tr>
<td>▶ Verify( (\sigma, m) ): ( \sigma^e = ? m \mod N )</td>
<td></td>
</tr>
</tbody>
</table>

About this threshold scheme:

SignKey \( d \) not recombined; can *reshare* \( d \) leaving \( e \) fixed; same \( \sigma \); efficient!

**Facilitating setting:** \( \exists \) dealer; \( \exists \) homomorphism; all parties learn \( m \).

**Not fault-tolerant:** a single sub-signer can boycott a correct signing.

Can other threshold schemes be implemented:

\( \not\exists \) dealer, \( \not\exists \) homomorphisms, secret-shared \( m \), withstanding \( f \) malicious signers?
# A simple example: RSA signature (or decryption) [RSA78]

<table>
<thead>
<tr>
<th><strong>Conventional scheme</strong> ((k = n = 1))</th>
<th><strong>A 3-out-of-3 threshold scheme</strong> ((k = n = 3))</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyGen (by signer):</td>
<td>KeyGen (by dealer):</td>
</tr>
<tr>
<td>- Public Modulus: (N = p \cdot q)</td>
<td>- Same (N, d, e)</td>
</tr>
<tr>
<td>- Secret SignKey: (d)</td>
<td>- SubKeys: (d_1, d_2, d_3 : d_1 + d_2 + d_3 = d \pmod{\phi})</td>
</tr>
<tr>
<td>- Public VerKey: (e = d^{-1} \pmod{\phi})</td>
<td>- Sign((m)): { separate: (s_i = m^{d_i} \pmod{N} : i = 1, 2, 3) combine: (\sigma = s_1 \cdot s_2 \cdot s_3 \pmod{N}) }</td>
</tr>
<tr>
<td>Sign((m)): (\sigma = m^d \pmod{N})</td>
<td>- Verify((\sigma, m)): (\sigma^e = ? m \pmod{N})</td>
</tr>
<tr>
<td>Verify((\sigma, m)): (\sigma^e = ? m \pmod{N})</td>
<td>- Verify((\sigma, m)): (\sigma^e = ? m \pmod{N})</td>
</tr>
</tbody>
</table>

**About this threshold scheme:**

SignKey \(d\) not recombined; can *reshare* \(d\) leaving \(e\) fixed; same \(\sigma\); efficient!

**Facilitating setting:** \(\exists\) dealer; \(\exists\) homomorphism; all parties learn \(m\).

**Not fault-tolerant:** a single sub-signer can boycott a correct signing.

**Can other threshold schemes be implemented:**

\(\nexists\) dealer, \(\nexists\) homomorphisms, secret-shared \(m\), withstanding \(f\) malicious signers?

**Yes**, using threshold cryptography
A simple example: RSA signature (or decryption) [RSA78]

Conventional scheme \((k = n = 1)\)

- KeyGen (by signer):
  - Public Modulus: \(N = p \cdot q\)
  - Secret SignKey: \(d\)
  - Public VerKey: \(e = d^{-1} \pmod{\phi}\)
- Sign \((m)\): \(\sigma = m^d \pmod{N}\)
- Verify \((\sigma, m)\): \(\sigma^e = m \pmod{N}\)

A 3-out-of-3 threshold scheme \((k = n = 3)\)

- KeyGen (by dealer):
  - Same \(N, d, e\)
  - SubKeys: \(d_1, d_2, d_3 : d_1 + d_2 + d_3 = d \pmod{\phi}\)
- Sign \((m)\): \(\sigma = s_1 \cdot s_2 \cdot s_3 \pmod{N}\)
- Verify \((\sigma, m)\): \(\sigma^e = m \pmod{N}\)

About this threshold scheme:

- SignKey \(d\) not recombined; can reshare \(d\) leaving \(e\) fixed; same \(\sigma\); efficient!

  Facilitating setting: \(\exists\) dealer; \(\exists\) homomorphism; all parties learn \(m\).

  Not fault-tolerant: a single sub-signer can boycott a correct signing.

Can other threshold schemes be implemented:

- \(\not\exists\) dealer, \(\not\exists\) homomorphisms, secret-shared \(m\), withstanding \(f\) malicious signers?

  Yes, using threshold cryptography (with more complicated schemes)
What do thresholds $k$ and $f$ mean?

2-out-of-3 signature:
▶ Availability: 2 nodes needed to sign ($k=2$, $f=1$)
▶ Key secrecy: okay while 2 shares are secret ($k=2$, $f=1$)
What do thresholds $k$ and $f$ mean?

**3-out-of-3 decryption:**

- **Availability:** 3 nodes needed to decrypt
- **Key secrecy:** okay while 1 share is secret
What do thresholds $k$ and $f$ mean?

3-out-of-3 decryption:

- **Availability**: 3 nodes needed to decrypt ($k = 3, f = 0$)
- **Key secrecy**: okay while 1 share is secret

But does any of these schemes improve security? (compared with a non-threshold scheme ($n = k = 1, f = 0$))

It depends: "$k$-out-of-$n$" or "$f$-out-of-$n$" is not a sufficient characterization for a comprehensive security assertion. Depends on attack model (e.g., attack surface, ...), system model (e.g., rejuvenations, ...), ...
What do thresholds $k$ and $f$ mean?

3-out-of-3 decryption:

- **Availability**: 3 nodes needed to decrypt ($k = 3$, $f = 0$)
- **Key secrecy**: okay while 1 share is secret ($k = 1$, $f = 2$)

But does any of these schemes improve security?

(compared with a non-threshold scheme ($n = k = 1$, $f = 0$))

It depends: "$k$-out-of-$n$" or "$f$-out-of-$n$" is not a sufficient characterization for a comprehensive security assertion. Depends on attack model (e.g., attack surface, ...), system model (e.g., rejuvenations, ...), ...
What do thresholds $k$ and $f$ mean?

3-out-of-3 decryption:

- **Availability**: 3 nodes needed to decrypt ($k = 3$, $f = 0$)
- **Key secrecy**: okay while 1 share is secret ($k = 1$, $f = 2$)

(Each security property has its own $k$ and $f$)
What do thresholds $k$ and $f$ mean?

3-out-of-3 decryption:

- **Availability**: 3 nodes needed to decrypt ($k = 3, f = 0$)
- **Key secrecy**: okay while 1 share is secret ($k = 1, f = 2$)

(Each security property has its own $k$ and $f$)

2-out-of-3 signature:

- **Availability**: 2 nodes needed to sign
- **Key secrecy**: okay while 2 shares are secret
What do thresholds $k$ and $f$ mean?

3-out-of-3 decryption:

- **Availability:** 3 nodes needed to decrypt ($k = 3$, $f = 0$)
- **Key secrecy:** okay while 1 share is secret ($k = 1$, $f = 2$)

(Each security property has its own $k$ and $f$)

2-out-of-3 signature:

- **Availability:** 2 nodes needed to sign ($k = 2$, $f = 1$)
- **Key secrecy:** okay while 2 shares are secret
What do thresholds $k$ and $f$ mean?

3-out-of-3 decryption:

- **Availability:** 3 nodes needed to decrypt ($k = 3$, $f = 0$)
- **Key secrecy:** okay while 1 share is secret ($k = 1$, $f = 2$)

(Each security property has its own $k$ and $f$)

2-out-of-3 signature:

- **Availability:** 2 nodes needed to sign ($k = 2$, $f = 1$)
- **Key secrecy:** okay while 2 shares are secret ($k = 2$, $f = 1$)
What do thresholds $k$ and $f$ mean?

3-out-of-3 decryption:

- **Availability**: 3 nodes needed to decrypt ($k = 3$, $f = 0$)
- **Key secrecy**: okay while 1 share is secret ($k = 1$, $f = 2$)

(Each security property has its own $k$ and $f$)

2-out-of-3 signature:

- **Availability**: 2 nodes needed to sign ($k = 2$, $f = 1$)
- **Key secrecy**: okay while 2 shares are secret ($k = 2$, $f = 1$)

But does any of these schemes improve security?
(compared with a non-threshold scheme ($n = k = 1$, $f = 0$))
What do thresholds $k$ and $f$ mean?

3-out-of-3 decryption:

- **Availability**: 3 nodes needed to decrypt ($k = 3, f = 0$)
- **Key secrecy**: okay while 1 share is secret ($k = 1, f = 2$)

(Each security property has its own $k$ and $f$)

2-out-of-3 signature:

- **Availability**: 2 nodes needed to sign ($k = 2, f = 1$)
- **Key secrecy**: okay while 2 shares are secret ($k = 2, f = 1$)

But does any of these schemes improve security?

(compared with a non-threshold scheme ($n = k = 1, f = 0$))

It depends: “$k$-out-of-$n$” or “$f$-out-of-$n$” is not a sufficient characterization for a comprehensive security assertion
What do thresholds $k$ and $f$ mean?

**3-out-of-3 decryption:**

- **Availability:** 3 nodes needed to decrypt ($k = 3$, $f = 0$)
- **Key secrecy:** okay while 1 share is secret ($k = 1$, $f = 2$)

(Each security property has its own $k$ and $f$)

**2-out-of-3 signature:**

- **Availability:** 2 nodes needed to sign ($k = 2$, $f = 1$)
- **Key secrecy:** okay while 2 shares are secret ($k = 2$, $f = 1$)

But does any of these schemes improve security? (compared with a non-threshold scheme ($n = k = 1, f = 0$))

It depends: “$k$-out-of-$n$” or “$f$-out-of-$n$” is not a sufficient characterization for a comprehensive security assertion

Depends on attack model (e.g., attack surface, ...), system model (e.g., rejuvenations, ...), ...
Outline

1. Introduction
2. Preliminaries
3. Step 1: NISTIR
4. Step 2: NTCW
5. Step 3: preliminary roadmap
6. Final remarks
NIST Internal Report (NISTIR) 8214

Challenges and Opportunities in Standardization and Validation of Threshold Cryptography.
NIST Internal Report (NISTIR) 8214

Threshold Schemes for Cryptographic Primitives — Challenges and Opportunities in Standardization and Validation of Threshold Cryptography. [BMV18] doi:10.6028/NIST.IR.8214

Threshold Schemes for Cryptographic Primitives — Challenges and Opportunities in Standardization and Validation of Threshold Cryptography. [BMV18] doi:10.6028/NIST.IR.8214

The report sets a basis for discussion:

- need to **characterize** threshold schemes
- need to **engage** with stakeholders
- need to **define** criteria for standardization

NIST Internal Report (NISTIR) 8214

Threshold Schemes for Cryptographic Primitives — Challenges and Opportunities in Standardization and Validation of Threshold Cryptography. [BMV18] doi:10.6028/NIST.IR.8214

The report sets a basis for discussion:

- need to characterize threshold schemes
- need to engage with stakeholders
- need to define criteria for standardization

Past timeline:

- 2018-July: Draft online 3 months for public comments
- 2018-October: Received comments from 13 external sources
- 2019-March: Final version online, along with “diff” and received comments

3. Step 1: NISTIR

Characterizing threshold schemes

To reflect on a threshold scheme, start by characterizing 4 main features:

• Kinds of threshold
• Communication interfaces
• Executing platform
• Setup and maintenance

Each feature spans distinct options that affect security in different ways. A characterization provides a better context for security assertions. But there are other factors...
Characterizing threshold schemes

To reflect on a threshold scheme, start by characterizing 4 main features:

- Kinds of threshold
- Executing platform
- Communication interfaces
- Setup and maintenance

The cliparts are from openclipart.org/detail/*, with * ∈ {71491, 190624, 101407, 161401, 161389}.
To reflect on a threshold scheme, start by characterizing 4 main features:

- Kinds of threshold
- Communication interfaces
- Executing platform
- Setup and maintenance

Each feature spans distinct options that affect security in different ways.
Characterizing threshold schemes

To reflect on a threshold scheme, start by characterizing **4 main features**: 

- Kinds of threshold
- Executing platform
- Communication interfaces
- Setup and maintenance

Each feature spans distinct options that affect security in different ways.

**A characterization provides a better context for security assertions.**
Characterizing threshold schemes

To reflect on a threshold scheme, start by characterizing **4 main features:**

- Kinds of threshold
- Communication interfaces
- Executing platform
- Setup and maintenance

Each feature spans distinct options that affect security in different ways.

A characterization provides a better context for security assertions.

But there are other factors ...
Deployment context
Deployment context

- **Application context.** Should it affect security requirements?
Deployment context

- **Application context.** Should it affect security requirements?
  - signature correctness — may be deferred to client
  - decryption correctness — may require *robust* protocol
3. Step 1: NISTIR

Deployment context

- **Application context.** Should it affect security requirements?
  - signature correctness — may be deferred to client
  - decryption correctness — may require *robust* protocol

- Conceivable attack types.
  - Active vs. passive
  - Static vs. adaptive
  - Stealth vs. detected
  - Invasive (physical) vs. non-invasive
  - Side-channel vs. communication interfaces
  - Parallel vs. sequential (wrt attacking nodes)
Deployment context

- **Application context.** Should it affect security requirements?
  - signature correctness — may be deferred to client
  - decryption correctness — may require *robust* protocol

- **Conceivable attack types.**
  - Active vs. passive
  - Static vs. adaptive
  - Stealth vs. detected
  - Invasive (physical) vs. non-invasive
  - Side-channel vs. communication interfaces
  - Parallel vs. sequential (wrt attacking nodes)

A threshold scheme *improving* security against an attack in an application *may be powerless or degrade* security for another attack in another application.
The validation challenge
The validation challenge

Devise standards of **testable and validatable** threshold schemes vs. devise **testing and validation for standardized** threshold schemes
The validation challenge

Devise standards of testable and validatable threshold schemes vs. devise testing and validation for standardized threshold schemes

Validation is needed in the federal context:

- need to use validated implementations [IC96] of standardized algorithms
- FIPS 140-2/3 defines, for cryptographic modules, 4 security levels: subsets of applicable security assertions [NIS01]

(FIPS = Federal Information Processing Standards)
Outline

1. Introduction
2. Preliminaries
3. Step 1: NISTIR
4. Step 2: NTCW
5. Step 3: preliminary roadmap
6. Final remarks
#NTCW2019

NIST Threshold Cryptography Workshop 2019

https://csrc.nist.gov/Events/2019/NTCW19
#NTCW2019

NIST Threshold Cryptography Workshop 2019

March 11–12, 2019 @
NIST Gaithersburg MD, USA

https://csrc.nist.gov/Events/2019/NTCW19
#NTCW2019

NIST Threshold Cryptography Workshop 2019

March 11–12, 2019 @ NIST Gaithersburg MD, USA

Countries (of affiliation) registered to the NIST Threshold Cryptography Workshop:
- United States: 75%
- Belgium: 9%
- Canada: 1%
- China: 1%
- Estonia: 4%
- France: 4%
- Israel: 1%
- Italy: 1%
- Switzerland: 2%
- Denmark: 2%

About 80 attendees

https://csrc.nist.gov/Events/2019/NTCW19
NIST Threshold Cryptography Workshop 2019

March 11–12, 2019 @
NIST Gaithersburg MD, USA

A platform for open interaction:

▶ hear about experiences with threshold crypto;
▶ get to know stakeholders;
▶ get input to reflect on roadmap and criteria.

https://csrc.nist.gov/Events/2019/NTCW19
Format and content

Accepted 15 external submissions:
- 2 panels
- 5 papers
- 8 presentations

Plus:
- 2 invited keynotes
- 4 NIST talks
- 2 feedback moments

Videos, papers and presentations online at the NTCW webpage: https://csrc.nist.gov/Events/2019/NTCW19

Discussion of diverse topics:
- threshold schemes in general (motivation and implementation feasibility);
- NIST standardization of cryptographic primitives
- a post-quantum threshold public-key encryption scheme;
- threshold signatures (adaptive security; elliptic curve digital signature algorithm);
- validation of cryptographic implementations;
- threshold circuit design (tradeoffs, pitfalls, combined attacks, verification tools);
- secret-sharing with leakage resilience;
- distributed symmetric-key encryption;
- applications and experience with threshold cryptography.
Format and content

Accepted 15 external submissions:

- 2 panels
- 5 papers
- 8 presentations

Videos, papers and presentations online at the NTCW webpage: https://csrc.nist.gov/Events/2019/NTCW19

Discussion of diverse topics:

- threshold schemes in general (motivation and implementation feasibility);
- NIST standardization of cryptographic primitives
- a post-quantum threshold public-key encryption scheme;
- threshold signatures (adaptive security; elliptic curve digital signature algorithm);
- validation of cryptographic implementations;
- threshold circuit design (tradeoffs, pitfalls, combined attacks, verification tools);
- secret-sharing with leakage resilience;
- distributed symmetric-key encryption;
- applications and experience with threshold cryptography.
Format and content

Accepted 15 external submissions:

- 2 panels
- 5 papers
- 8 presentations

Plus:

- 2 invited keynotes
- 4 NIST talks
- 2 feedback moments

Videos, papers and presentations online at the NTCW webpage:
https://csrc.nist.gov/Events/2019/NTCW19

Discussion of diverse topics:

- threshold schemes in general (motivation and implementation feasibility);
- NIST standardization of cryptographic primitives
- a post-quantum threshold public-key encryption scheme;
- threshold signatures (adaptive security; elliptic curve digital signature algorithm);
- validation of cryptographic implementations;
- threshold circuit design (tradeoffs, pitfalls, combined attacks, verification tools);
- secret-sharing with leakage resilience;
- distributed symmetric-key encryption;
- applications and experience with threshold cryptography.
Format and content

Accepted 15 external submissions:

- 2 panels
- 5 papers
- 8 presentations

Plus:

- 2 invited keynotes
- 4 NIST talks
- 2 feedback moments

Videos, papers and presentations online at the NTCW webpage: https://csrc.nist.gov/Events/2019/NTCW19
Format and content

Accepted 15 external submissions:
- 2 panels
- 5 papers
- 8 presentations

Plus:
- 2 invited keynotes
- 4 NIST talks
- 2 feedback moments

Videos, papers and presentations online at the NTCW webpage: https://csrc.nist.gov/Events/2019/NTCW19

Discussion of diverse topics:
- threshold schemes in general (motivation and implementation feasibility);
- NIST standardization of cryptographic primitives
- a post-quantum threshold public-key encryption scheme;
- threshold signatures (adaptive security; elliptic curve digital signature algorithm);
- validation of cryptographic implementations;
- threshold circuit design (tradeoffs, pitfalls, combined attacks, verification tools);
- secret-sharing with leakage resilience;
- distributed symmetric-key encryption;
- applications and experience with threshold cryptography.
4. Step 2: NTCW

Results

A step in driving an open and transparent process towards standardization of threshold schemes for cryptographic primitives. (See NISTIR 7977)

Some notes:
▶ differences in granularity (building blocks vs. full functionalities);
▶ separation of single-device vs. multi-party;
▶ importance of envisioning applications;
▶ stakeholders' willingness to contribute;
▶ usefulness of explaining rationale (e.g., as complimented for the NISTIR);
▶ encouragement to move forward.

These elements are helpful for the next step... designing a roadmap
Results

A step in driving an open and transparent process towards standardization of threshold schemes for cryptographic primitives. (See NISTIR 7977)
4. Step 2: NTCW

Results

A step in *driving an open and transparent process towards standardization* of *threshold schemes for cryptographic primitives*. *(See NISTIR 7977)*

Some notes:

- differences in granularity (building blocks vs. full functionalities);
- separation of single-device vs. multi-party;
- importance of envisioning applications;
- stakeholders’ willingness to contribute;
- usefulness of explaining rationale (e.g., as complimented for the NISTIR);
- encouragement to move forward.
Results

A step in *driving an open and transparent process towards standardization* of *threshold schemes for cryptographic primitives*. (See NISTIR 7977)

Some notes:

- differences in granularity (building blocks vs. full functionalities);
- separation of single-device vs. multi-party;
- importance of envisioning applications;
- stakeholders’ willingness to contribute;
- usefulness of explaining rationale (e.g., as complimented for the NISTIR);
- encouragement to move forward.

These elements are helpful for the next step ... designing a roadmap
Outline

1. Introduction
2. Preliminaries
3. Step 1: NISTIR
4. Step 2: NTCW
5. Step 3: preliminary roadmap
6. Final remarks
Preliminary roadmap (ongoing)

We are writing a draft “preliminary roadmap”
Preliminary roadmap (ongoing)

We are writing a draft “preliminary roadmap”
(getting a map; deciding where to go; thinking how to get there)
Preliminary roadmap (ongoing)

We are writing a draft “preliminary roadmap”
(getting a map; deciding where to go; thinking how to get there)

Need: **mapping layers** (coordinates) and **weighing factors**
Preliminary roadmap (ongoing)

We are writing a draft “preliminary roadmap”
(getting a map; deciding where to go; thinking how to get there)

Need: *mapping layers* (coordinates) and *weighing factors*

Disclaimer: the structure suggested in the next slides is still subject to change.
Mapping layers

An abstract layered decomposition of the threshold standardization space

Four layers

<table>
<thead>
<tr>
<th>Standardization space for threshold schemes for cryptographic primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Route A: simple thresholdization</td>
</tr>
<tr>
<td>▶ Route B: compositional designs</td>
</tr>
<tr>
<td>▶ Route C: new primitives</td>
</tr>
<tr>
<td>▶ Route D: gadgets</td>
</tr>
</tbody>
</table>
Mapping layers

An abstract layered decomposition of the threshold standardization space

Four layers: **domains**

- Single-device (domain)
- Multi-party (domain)

Standardization space for threshold schemes for cryptographic primitives

- Route A: simple thresholdization
- Route B: compositional designs
- Route C: new primitives
- Route D: gadgets
Mapping layers

An abstract layered decomposition of the threshold standardization space

Four layers: domains, routes

- Single-device (domain)
  - Route A
  - Route B
  - Route C

- Multi-party (domain)
  - Route A
  - Route B
  - Route C

- Route A: simple thresholdization
- Route B: compositional designs
- Route C: new primitives
Mapping layers

An abstract layered decomposition of the threshold standardization space

Four layers: domains, routes

- Single-device (domain)
  - Route A
  - Route B
  - Route C

- Multi-party (domain)
  - Route A
  - Route B
  - Route C

- Route D: gadgets

- Route A: simple thresholdization
- Route B: compositional designs
- Route C: new primitives
Mapping layers

An abstract layered decomposition of the threshold standardization space

Four layers: domains, routes, **primitives**

<table>
<thead>
<tr>
<th>Single-device (domain)</th>
<th>Multi-party (domain)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Route A</td>
<td>Route A</td>
</tr>
<tr>
<td>Route B</td>
<td>Route B</td>
</tr>
<tr>
<td>Route C</td>
<td>Route C</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Route A: simple thresholdization
- Route B: compositional designs
- Route C: new primitives
- Route D: gadgets

Standardization space for threshold schemes for cryptographic primitives

- Primitive 1
- Primitive $n$
Mapping layers

An abstract layered decomposition of the threshold standardization space

Four layers: domains, routes, primitives, **modes**

- **Single-device (domain)**
  - Route A
  - Route B
  - Route C

- **Multi-party (domain)**
  - Route A
  - Route B
  - Route C
  - Route D

- **Primitives**
  - Primitive 1
  - Primitive n

- **Modes**
  - Mode 1
  - Mode m

- Route A: simple thresholdization
- Route B: compositional designs
- Route C: new primitives
- Route D: gadgets
Some conceived examples

Primitives across routes:

- A: RSA decryption & signature; Schnorr signature; ECC key-gen; AES (single-device) threshold circuit design against leakage.
- B: ECDSA signature; RSA key-gen; AES enciphering; AES (single-device) threshold circuit against combined attacks.
- C: post-quantum signing & decryption; lightweight-crypto threshold.
- D: secret sharing; distributed RNG; consensus.

Modes:

- threshold signature with secret-shared key vs. multi-signature (independent keys);
- operation on secret-shared plaintext;
- honest majority; robust with fault detection;
- asynchronous environment.

Not every possible combination needs to be a standardization goal.
Some conceived examples

**Primitives across routes:**

- **A:** RSA decryption & signature; Schnorr signature; ECC key-gen; AES (single-device) threshold circuit design against leakage.

- **B:**

- **C:**

- **D:**

**Modes:**
Some conceived examples

**Primitives across routes:**

- **A:** RSA decryption & signature; Schnorr signature; ECC key-gen; AES (single-device) threshold circuit design against leakage.
- **B:** ECDSA signature; RSA key-gen; AES enciphering; AES (single-device) threshold circuit against combined attacks.
- **C:**
- **D:**

**Modes:**
Some conceived examples

*Primitives across routes:*

- **A:** RSA decryption & signature; Schnorr signature; ECC key-gen; AES (single-device) threshold circuit design against leakage.
- **B:** ECDSA signature; RSA key-gen; AES enciphering; AES (single-device) threshold circuit against combined attacks.
- **C:** post-quantum signing & decryption; lightweight-crypto threshold.
- **D:**

*Modes:*

Some conceived examples

**Primitives across routes:**

- **A:** RSA decryption & signature; Schnorr signature; ECC key-gen; AES (single-device) threshold circuit design against leakage.
- **B:** ECDSA signature; RSA key-gen; AES enciphering; AES (single-device) threshold circuit against combined attacks.
- **C:** post-quantum signing & decryption; lightweight-crypto threshold.
- **D:** secret sharing; distributed RNG; consensus.

**Modes:**
Some conceived examples

**Primitives across routes:**

- **A:** RSA decryption & signature; Schnorr signature; ECC key-gen; AES (single-device) threshold circuit design against leakage.
- **B:** ECDSA signature; RSA key-gen; AES enciphering; AES (single-device) threshold circuit against combined attacks.
- **C:** post-quantum signing & decryption; lightweight-crypto threshold.
- **D:** secret sharing; distributed RNG; consensus.

**Modes:**

- threshold signature with secret-shared key vs. multi-signature (independent keys);
Some conceived examples

**Primitives across routes:**

- **A:** RSA decryption & signature; Schnorr signature; ECC key-gen; AES (single-device) threshold circuit design against leakage.
- **B:** ECDSA signature; RSA key-gen; AES enciphering; AES (single-device) threshold circuit against combined attacks.
- **C:** post-quantum signing & decryption; lightweight-crypto threshold.
- **D:** secret sharing; distributed RNG; consensus.

**Modes:**

- threshold signature with secret-shared key vs. multi-signature (independent keys);
- operation on secret-shared plaintext;
Some conceived examples

**Primitives across routes:**
- **A:** RSA decryption & signature; Schnorr signature; ECC key-gen; AES (single-device) threshold circuit design against leakage.
- **B:** ECDSA signature; RSA key-gen; AES enciphering; AES (single-device) threshold circuit against combined attacks.
- **C:** post-quantum signing & decryption; lightweight-crypto threshold.
- **D:** secret sharing; distributed RNG; consensus.

**Modes:**
- threshold signature with secret-shared key vs. multi-signature (independent keys);
- operation on secret-shared plaintext;
- honest majority; robust with fault detection;
Some conceived examples

**Primitives across routes:**

- **A:** RSA decryption & signature; Schnorr signature; ECC key-gen; AES (single-device) threshold circuit design against leakage.
- **B:** ECDSA signature; RSA key-gen; AES enciphering; AES (single-device) threshold circuit against combined attacks.
- **C:** post-quantum signing & decryption; lightweight-crypto threshold.
- **D:** secret sharing; distributed RNG; consensus.

**Modes:**

- threshold signature with secret-shared key vs. multi-signature (independent keys);
- operation on secret-shared plaintext;
- honest majority; robust with fault detection;
- asynchronous environment.
Some conceived examples

**Primitives across routes:**

- **A:** RSA decryption & signature; Schnorr signature; ECC key-gen; AES (single-device) threshold circuit design against leakage.
- **B:** ECDSA signature; RSA key-gen; AES enciphering; AES (single-device) threshold circuit against combined attacks.
- **C:** post-quantum signing & decryption; lightweight-crypto threshold.
- **D:** secret sharing; distributed RNG; consensus.

**Modes:**

- threshold signature with secret-shared key vs. multi-signature (independent keys);
- operation on secret-shared plaintext;
- honest majority; robust with fault detection;
- asynchronous environment.

Not every possible combination needs to be a standardization goal
Weighing factors

The four layers provide a map. But where to look in the map?
Weighing factors

The four layers provide a map. But where to look in the map?

- **Application motivations:**
  - threshold circuit design in single-device (address side-channel leakage)
  - distribute trust across several operators of crypto primitives
  - multi-signatures in crypto currencies
  - privacy preserving modes (e.g., secret-shared plaintext)

- **Useful features:**
  - efficiency and practicality
  - suitability for automated testing
  - ability to rejuvenate components
Weighing factors

The four layers provide a map. But where to look in the map?

- **Application motivations:**
  - threshold circuit design in single-device (address side-channel leakage)
  - distribute trust across several operators of crypto primitives*
  - multi-signatures in crypto currencies
  - privacy preserving modes (e.g., secret-shared plaintext)
  - ...

* (emphasis on approved conventional primitives)

- **Useful features:**
Weighing factors

The four layers provide a map. But where to look in the map?

► **Application motivations:**
  ► threshold circuit design in single-device (address side-channel leakage)
  ► distribute trust across several operators of crypto primitives*
  ► multi-signatures in crypto currencies
  ► privacy preserving modes (e.g., secret-shared plaintext)
  ► ...
  *(emphasis on approved conventional primitives)*

► **Useful features:**
  ► efficiency and practicality
  ► suitability for automated testing
  ► ability to rejuvenate components
  ► ...

*(emphasis on approved conventional primitives)*
5. Step 3: preliminary roadmap

Hereafter

Soon:
- Draft “preliminary roadmap” asking feedback, e.g., on:
  - elements within layers, application motivations and other factors
  - primitives/modes to focus on (and respective security properties)
  - possible elements to adopt/adapt from other standards

Later:
- separate criteria for separate focuses; calls for contributions
  - Example routes for calls for contributions:
    - algorithms for standardization
    - reference implementations and comparisons
    - research contributions
    - ...
  - Possibly fit some of these in a 2nd workshop (??)
Soon: Draft “preliminary roadmap” asking feedback
Hereafter

**Soon:** Draft “preliminary roadmap” asking feedback, e.g., on:
- elements within *layers, application motivations and other factors*
- primitives/modes to focus on (and respective security properties)
- possible elements to adopt/adapt from other standards
Hereafter

Soon: Draft “preliminary roadmap” asking feedback, e.g., on:
▶ elements within *layers, application motivations* and *other factors*
▶ primitives/modes to focus on (and respective security properties)
▶ possible elements to adopt/adapt from other standards

Later: separate criteria for separate focuses; calls for contributions
Hereafter

Soon: Draft “preliminary roadmap” asking feedback, e.g., on:
▶ elements within layers, application motivations and other factors
▶ primitives/modes to focus on (and respective security properties)
▶ possible elements to adopt/adapt from other standards

Later: separate criteria for separate focuses; calls for contributions

Example *routes* for calls for contributions:
▶ algorithms for standardization
▶ reference implementations and comparisons
▶ research contributions
▶ ...

Possibly fit some of these in a 2\textsuperscript{nd} workshop (?)
Outline

1. Introduction
2. Preliminaries
3. Step 1: NISTIR
4. Step 2: NTCW
5. Step 3: preliminary roadmap
6. Final remarks
Final remarks

Threshold schemes have potential to address single-points of failure:

- In technology when crypto implementations have vulnerabilities
- At the human level when crypto operators go rogue

There exist numerous researched threshold schemes

It is time to move towards (some) standardization

We would like to have a process in collaboration with stakeholders!
Final remarks

- Threshold schemes have potential to address single-points of failure:
  - in technology ... when crypto implementations have vulnerabilities
  - at the human level ... when crypto operators go rogue
Final remarks

- Threshold schemes have potential to address single-points of failure:
  - in technology ... when crypto implementations have vulnerabilities
  - at the human level ... when crypto operators go rogue

- There exist numerous researched threshold schemes
Final remarks

- Threshold schemes have potential to address single-points of failure:
  - in technology ... when crypto implementations have vulnerabilities
  - at the human level ... when crypto operators go rogue

- There exist numerous researched threshold schemes

- It is time to move towards (some) standardization
Final remarks

- Threshold schemes have potential to address single-points of failure:
  - in technology ... when crypto implementations have vulnerabilities
  - at the human level ... when crypto operators go rogue

- There exist numerous researched threshold schemes

- It is time to move towards (some) standardization

We would like to have a process in collaboration with stakeholders!
Project webpage: https://csrc.nist.gov/Projects/Threshold-Cryptography
Project email adress: threshold-crypto@nist.gov
NISTIR 8214: https://csrc.nist.gov/publications/detail/nistir/8214/final
NTCW webpage: https://csrc.nist.gov/Events/2019/NTCW19
Forum: https://groups.google.com/a/list.nist.gov/forum/#!forum/tc-forum
   (register for announcements; we can add your email if you send us a request)
6. Final remarks

Thank you for your attention.

Project webpage: https://csrc.nist.gov/Projects/Threshold-Cryptography
Project email address: threshold-crypto@nist.gov
NISTIR 8214: https://csrc.nist.gov/publications/detail/nistir/8214/final
NTCW webpage: https://csrc.nist.gov/Events/2019/NTCW19
Forum: https://groups.google.com/a/list.nist.gov/forum/#!forum/tc-forum
(register for announcements; we can add your email if you send us a request)

Word cloud based on the NISTIR 8214 Presentation at the International Cryptographic Module Conference May 16, 2019 @ Vancouver, Canada

Disclaimer.
Opinions expressed in this presentation are from the author(s) and are not to be construed as official or as views of the U.S. Department of Commerce. The identification of any commercial product or trade names in this presentation does not imply endorsement of recommendation by NIST, nor is it intended to imply that the material or equipment identified are necessarily the best available for the purpose.

Disclaimer.
Some external-source images and cliparts were included/adapted in this presentation with the expectation of such use constituting licensed and/or fair use.
Thank you for your attention

- Project webpage: https://csrc.nist.gov/Projects/Threshold-Cryptography
- Project email address: threshold-crypto@nist.gov
- NTCW webpage: https://csrc.nist.gov/Events/2019/NTCW19
- Forum: https://groups.google.com/a/list.nist.gov/forum/#!forum/tc-forum
  (register for announcements; we can add your email if you send us a request)

Presentation at the International Cryptographic Module Conference
May 16, 2019 @ Vancouver, Canada
luis.brandao@nist.gov
Thank you for your attention

- Project webpage: https://csrc.nist.gov/Projects/Threshold-Cryptography
- Project email address: threshold-crypto@nist.gov
- NTCW webpage: https://csrc.nist.gov/Events/2019/NTCW19
- Forum: https://groups.google.com/a/list.nist.gov/forum/#!forum/tc-forum
  (register for announcements; we can add your email if you send us a request)

Presentation at the International Cryptographic Module Conference
May 16, 2019 @ Vancouver, Canada
luis.brandao@nist.gov

Disclaimer. Opinions expressed in this presentation are from the author(s) and are not to be construed as official or as views of the U.S. Department of Commerce. The identification of any commercial product or trade names in this presentation does not imply endorsement of recommendation by NIST, nor is it intended to imply that the material or equipment identified are necessarily the best available for the purpose.

Disclaimer. Some external-source images and cliparts were included/adapted in this presentation with the expectation of such use constituting licensed and/or fair use.
References


Extra slides

Next follow some extra slides
Reliability ($R$) — one metric of security

Probability that a security property (e.g., secrecy) never fails during a mission time

Time normalized: $\tau = 1$ is the expected time to failure (ETTF) of a node
Reliability ($\mathcal{R}$) — one metric of security

Probability that a security property (e.g., secrecy) never fails during a mission time

**A possible model:** each node fails (independently) with constant rate probability

<table>
<thead>
<tr>
<th>Curve</th>
<th>$\mathcal{R}$ of key-secrecy in a</th>
<th>$n$</th>
<th>$f$</th>
<th>$\tau_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-out-of-1 sig-scheme</td>
<td>$\triangle$</td>
<td>1</td>
<td>0</td>
<td>—</td>
</tr>
</tbody>
</table>

[BB12] Time normalized: $\tau = 1$ is the expected time to failure (ETTF) of a node
Reliability ($\mathcal{R}$) — one metric of security

Probability that a security property (e.g., secrecy) never fails during a mission time

A possible model: each node fails (independently) with constant rate probability

<table>
<thead>
<tr>
<th>Curve</th>
<th>$\mathcal{R}$ of key-secrecy in a $n_f$ sig-scheme</th>
<th>$n$</th>
<th>$f$</th>
<th>$\tau_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-out-of-1 sig-scheme</td>
<td>—</td>
<td>1</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>2-out-of-3 sig-scheme</td>
<td>—</td>
<td>3</td>
<td>1</td>
<td>0.693</td>
</tr>
</tbody>
</table>

$\tau_{\text{max}} = \max \left( t : \mathcal{R}_f(t) > \mathcal{R}_0(t) \right)$

Time normalized: $\tau = 1$ is the expected time to failure (ETTF) of a node

[BB12]
Reliability ($\mathcal{R}$) — one metric of security

Probability that a security property (e.g., secrecy) never fails during a mission time

A possible model: each node fails (independently) with constant rate probability

![Graph showing reliability over time](image)

<table>
<thead>
<tr>
<th>Curve</th>
<th>$\mathcal{R}$ of key-secrecy in a</th>
<th>$n$</th>
<th>$f$</th>
<th>$\tau_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>△</td>
<td>1-out-of-1 sig-scheme</td>
<td>1</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>⬇</td>
<td>2-out-of-3 sig-scheme</td>
<td>3</td>
<td>1</td>
<td>0.693</td>
</tr>
</tbody>
</table>

$\tau_{\text{max}} = \max \left( t : \mathcal{R}_f^n(t) > \mathcal{R}_0^1(t) \right)$

[BB12] Time normalized: $\tau = 1$ is the expected time to failure (ETTF) of a node

Reliability can be degraded when increasing the fault-tolerance threshold $f$. 
Reliability ($\mathcal{R}$) — one metric of security

Probability that a security property (e.g., secrecy) never fails during a mission time

**A possible model:** each node fails (independently) with constant rate probability

![Graph showing reliability ($\mathcal{R}$) against time ($\tau$).]

**Table:**

<table>
<thead>
<tr>
<th>Curve</th>
<th>$\mathcal{R}$ of key-secrecy in a</th>
<th>$n$</th>
<th>$f$</th>
<th>$\tau_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-out-of-1 sig-scheme</td>
<td>$\triangle$</td>
<td>1</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>2-out-of-3 sig-scheme</td>
<td>$\circ$</td>
<td>3</td>
<td>1</td>
<td>0.693</td>
</tr>
</tbody>
</table>

$\tau_{\text{max}} = \max(t : \mathcal{R}_f^n(t) > \mathcal{R}_0^1(t))$

[BB12] Time normalized: $\tau = 1$ is the expected time to failure (ETTF) of a node

Reliability can be degraded when increasing the fault-tolerance threshold $f$

Note: rejuvenation of nodes can attenuate the reliability-degradation
Another model

What if all nodes are compromised (e.g., leaky) from the start?
Another model

What if all nodes are compromised (e.g., leaky) from the start?

Threshold scheme may still be effective, if it increases the cost of exploitation!

(e.g., if exploiting a leakage vulnerability requires exponential number of traces for high-order Differential Power Analysis)
Another model

What if all nodes are compromised (e.g., leaky) from the start?

Threshold scheme may still be effective, if it increases the cost of exploitation!

(e.g., if exploiting a leakage vulnerability requires exponential number of traces for high-order Differential Power Analysis)

Challenge questions:

▶ which models are realistic / match state-of-the-art attacks?
▶ what concrete parameters (e.g., $n$) thwart real attacks?
Two hints
Two hints

Robust $k$-out-of-$n$ Threshold RSA Signature \cite{Sho00}

\[\text{Works iff } \geq k \text{ parties are available:} \]

- homomorphism allows combining (slightly tweaked) sub-signatures.
- Robust: sub-signers prove (efficient NIZKP) correct sub-signatures. (NIZK = non-interactive zero-knowledge proof of knowledge)

Threshold Schnorr (multi-)signature \cite{BN06}

- Different public key per signer $\rightarrow$ no dealer, dynamic signer-set
- Verifier decides the threshold and knows who signed
- DL-based homomorphism $\rightarrow$ size equal to 1 signature (DL = Discrete-Logarithm)
Two hints

Robust \( k \)-out-of-\( n \) Threshold RSA Signature \([\text{Sho00}]\)

- Works iff \( \geq k \) parties are available: homomorphism allows combining (slightly tweaked) sub-signatures.
Two hints

Robust $k$-out-of-$n$ Threshold RSA Signature [Sho00]

- Works iff $\geq k$ parties are available: homomorphism allows combining (slightly tweaked) sub-signatures.

- Robust: sub-signers prove (efficient NIZKP) correct sub-signatures.

(NIZK = non-interactive zero-knowledge proof of knowledge)
Two hints

Robust $k$-out-of-$n$ Threshold RSA Signature [Sho00]
  - Works iff $\geq k$ parties are available: homomorphism allows combining (slightly tweaked) sub-signatures.
  - Robust: sub-signers prove (efficient NIZKP) correct sub-signatures.

Threshold Schnorr (multi-)signature [BN06]
Two hints

Robust $k$-out-of-$n$ Threshold RSA Signature \cite{Sho00}

- Works iff $\geq k$ parties are available: homomorphism allows combining (slightly tweaked) sub-signatures.
- Robust: sub-signers prove (efficient NIZKP) correct sub-signatures.

(NIZK = non-interactive zero-knowledge proof of knowledge)

Threshold Schnorr (multi-)signature \cite{BN06}

- Different public key per signer $\rightarrow$ no dealer, dynamic signer-set
Two hints

**Robust** $k$-out-of-$n$ Threshold RSA Signature [Sho00]

- Works iff $\geq k$ parties are available: homomorphism allows combining (slightly tweaked) sub-signatures.
- Robust: sub-signers prove (efficient NIZKP) correct sub-signatures.

(NIZK = non-interactive zero-knowledge proof of knowledge)

**Threshold Schnorr (multi-)signature** [BN06]

- Different public key per signer $\rightarrow$ no dealer, dynamic signer-set
- Verifier decides the threshold and knows who signed
Two hints

Robust $k$-out-of-$n$ Threshold RSA Signature [Sho00]
- Works iff $\geq k$ parties are available: homomorphism allows combining (slightly tweaked) sub-signatures.
- Robust: sub-signers prove (efficient NIZKP) correct sub-signatures.

(NIZK = non-interactive zero-knowledge proof of knowledge)

Threshold Schnorr (multi-)signature [BN06]
- Different public key per signer $\rightarrow$ no dealer, dynamic signer-set
- Verifier decides the threshold and knows who signed
- DL-based homomorphism $\rightarrow$ size equal to 1 signature

(DL = Discrete-Logarithm)
A DL-based example: threshold Schnorr signature

(DL = Discrete-Logarithm)
A DL-based example: threshold Schnorr signature
(DL = Discrete-Logarithm)

(Next: ignore details — just making comparative remarks)
A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

Non-threshold scheme [Sch90]

- Space: $G, g$ (group, generator)
- KeyGen (by signer):
  - Secret SignKey: $x \in \mathbb{Z}_q$
  - Public VerKey: $X = g^{-x}$
- $\text{Sign}_x(m)$ by signer:
  - $R = g^r$
  - $c = q \cdot H(R|m)$
  - $s = q \cdot r + x \cdot c$
  - output $\sigma = (s, c)$
- $\text{Verify}_X(\sigma, m)$:
  - calculate $R = g^s X^c$
  - check $H(R|m) = c$

A multi-signature scheme [BN06]

- Space: same $G, g$
- KeyGen (by parties $i = 1, ..., n$):
  - Secret SignKey: $x_i \in \mathbb{Z}_q$
  - Public VerKey: $X_i = g^{x_i}$
- $\text{Sign}_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i = q \cdot H(X_i|R||I|m)$
  - $s = q \sum_{i \in L} s_i = \sum_{i \in I} (r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- $\text{Verify}(\sigma, m)$:
  - calculate $c_i = H(X_i|R|M||I|m)$
  - check $g^s = R \prod_{i \in I} X_i^{c_i}$

Some features:
- no dealer;
- dynamic threshold (verifier decides what is acceptable);
- dynamic set of signers;
- verifying $\Rightarrow$ knowing who signed.
A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

**Non-threshold scheme** [Sch90]

- **Space:** $G, g$ (group, generator)
- **KeyGen (by signer):**
  - **Secret SignKey:** $x \in \mathbb{Z}_q$
  - **Public VerKey:** $X = g^{-x}$
- **Sign$_x(m)$ by signer:**
  - $R = g^r$
  - $c = H(R||m)$
  - $s = r + x \cdot c$
  - output $\sigma = (s, c)$
- **Verify$_x(\sigma, m)$:**
  - calculate $R = g^sX^c$
  - check $H(R||m) = c$

**A multi-signature scheme** [BN06]

- **Space:** same $G, g$
- **KeyGen (by parties $i = 1, \ldots, n$):**
  - **Secret SignKey:** $x_i \in \mathbb{Z}_q$
  - **Public VerKey:** $X_i = g^{x_i}$
- **Sign$_{x,L}(m)$ by subset $I \subseteq \{1, \ldots, n\}$**
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i = H(X_i||R||I||m)$
  - $s = \sum_{i \in I} s_i = \sum_{i \in I} (r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- **Verify$(\sigma, m)$:**
  - calculate $c_i = H(X_i||R||M||I||m)$
  - check $g^s = R \prod_{i \in I} X_i^{c_i}$

Some features:
- no dealer;
- dynamic threshold (verifier decides what is acceptable);
- dynamic set of signers;
- verifying $\Rightarrow$ knowing who signed.
A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

Non-threshold scheme [Sch90]

- **Space**: \(G, g\) (group, generator)
- **KeyGen** (by signer):
  - Secret SignKey: \(x \in \mathbb{Z}_q\)
  - Public VerKey: \(X = g^{-x}\)
- **Sign** \(_x(m)\) by signer:
  - \(R = g^r\)
  - \(c = H(R|m)\)
  - \(s = r + x \cdot c\)
  - output \(\sigma = (s, c)\)
- **Verify** \(_x(\sigma, m)\):
  - calculate \(R = g^sX^c\)
  - check \(H(R|m) = c\)

A multi-signature scheme [BN06]

- **Space**: same \(G, g\)
- **KeyGen** (by parties \(i = 1, \ldots, n\)):
  - Secret SignKey: \(x_i \in \mathbb{Z}_q\)
  - Public VerKey: \(X_i = g^{x_i}\)
- **Sign** \(_{x,L}(m)\) by subset \(I \subseteq \{1, \ldots, n\}\):
  - \(R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}\)
  - \(c_i = H(X_i|R|I|m)\)
  - \(s = \sum_{i \in I} s_i = \sum_{i \in I} (r_i + x_i c_i)\)
  - output \(\sigma = (R, s)\)
- **Verify** \((\sigma, m)\):
  - calculate \(c_i = H(X_i|R|M|I|m)\)
  - check \(g^s = \prod_{i \in I} X_i^{c_i}\)

Some features:
- no dealer;
- dynamic threshold (verifier decides what is acceptable);
- dynamic set of signers;
- verifying \(\Rightarrow\) knowing who signed.
A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

**Non-threshold scheme** [Sch90]

- **Space**: $G, g$ (group, generator)
- **KeyGen** *(by signer)*:
  - Secret SignKey: $x \in \mathbb{Z}_q$
  - Public VerKey: $X = g^{-x}$
- **Sign$_x(m)$** by signer:
  - $R = g^r$
  - $c = H(R||m)$
  - $s = r + x \cdot c$
  - output $\sigma = (s, c)$
- **Verify$_X(\sigma, m)$**:
  - calculate $R = g^s X^c$
  - check $H(R||m) = ? c$

**A multi-signature scheme** [BN06]

- **Space**: same $G, g$
- **KeyGen** *(by parties $i = 1, ..., n$)*:
  - Secret SignKey: $x_i \in \mathbb{Z}_q$
  - Public VerKey: $X_i = g^{x_i}$
- **Sign$_{x,I}(m)$** by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i = H(X_i||R||I||m)$
  - $s = \sum_{i \in I} s_i = \sum_{i \in I} (r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- **Verify$(\sigma, m)$**:
  - calculate $c_i = H(X_i||R||M||I||m)$
  - check $g^s = R \prod_{i \in I} X_i^{c_i}$

Some features:
- no dealer;
- dynamic threshold (verifier decides what is acceptable);
- dynamic set of signers;
- verifying $\Rightarrow$ knowing who signed.
A DL-based example: threshold Schnorr signature
(DL = Discrete-Logarithm)

(Next: ignore details — just making comparative remarks)

**Non-threshold scheme** \([\text{Sch90}]\)

- **Space:** \(G, g\) (group, generator)
- **KeyGen (by signer):**
  - Secret SignKey: \(x \in \mathbb{Z}_q\)
  - Public VerKey: \(X = g^{-x}\)
- **Sign\(_x\)(m)** by signer:
  - \(R = g^r\)
  - \(c = _q H(R||m)\)
  - \(s = _q r + x \cdot c\)
  - output \(\sigma = (s, c)\)
- **Verify\(_x\)(\(\sigma, m)\):**
  - calculate \(R = g^sX^c\)
  - check \(H(R||m) = ? c\)

**A multi-signature scheme** \([\text{BN06}]\)

- **Space:** same \(G, g\)
- **KeyGen (by parties \(i = 1, \ldots, n\)):**
  - Secret SignKey: \(x_i \in \mathbb{Z}_q\)
  - Public VerKey: \(X_i = g^{x_i}\)
- **Sign\(_{x,L}(m)\)** by subset \(I \subseteq \{1, \ldots, n\}\)
  - \(R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}\)
  - \(c_i = _q H(X_i||R||I||m)\)
  - \(s = _q \sum_{i \in L} s_i = \sum_{i \in I} (r_i + x_i c_i)\)
  - output \(\sigma = (R, s)\)
- **Verify(\(\sigma, m)\):**
  - calculate \(c_i = H(X_i||R||M||I||m)\)
  - check \(g^s = ? R \prod_{i \in I} X_i^{c_i}\)

Some features:
- no dealer;
- dynamic threshold (verifier decides what is acceptable);
- dynamic set of signers;
- verifying ⇒ knowing who signed.
A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

Non-threshold scheme [Sch90]

- Space: $G, g$ (group, generator)
- KeyGen (by signer):
  - Secret SignKey: $x \in \mathbb{Z}_q$
  - Public VerKey: $X = g^{-x}$
- $\text{Sign}_x(m)$ by signer:
  - $R = g^r$
  - $c = q \cdot H(R || m)$
  - $s = q \cdot r + x \cdot c$
  - output $\sigma = (s, c)$
- Verify$_X(\sigma, m)$:
  - calculate $R = g^s X^c$
  - check $H(R || m) = c$

A multi-signature scheme [BN06]

- Space: same $G, g$
- KeyGen (by parties $i = 1, \ldots, n$):
  - Secret SignKey: $x_i \in \mathbb{Z}_q$
  - Public VerKey: $X_i = g^{x_i}$
- $\text{Sign}_{x, I}(m)$ by subset $I \subseteq \{1, \ldots, n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i = q \cdot H(X_i || R || I || m)$
  - $s = q \cdot \sum_{i \in I} s_i = \sum_{i \in I} (r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify$(\sigma, m)$:
  - calculate $c_i = H(X_i || R || M || I || m)$
  - check $g^s = R \prod_{i \in I} X_i^{c_i}$

Some features:
- no dealer;
- dynamic threshold (verifier decides what is acceptable);
- dynamic set of signers;
- verifying $\Rightarrow$ knowing who signed.
A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

Non-threshold scheme [Sch90]

- Space: $G, g$ (group, generator)
- KeyGen (by signer):
  - Secret SignKey: $x \in \mathbb{Z}_q$
  - Public VerKey: $X = g^{-x}$
- Sign$_x(m)$ by signer:
  - $R = g^r$
  - $c = qH(R|m)$
  - $s = q(r + x \cdot c)$
  - output $\sigma = (s, c)$
- Verify$_X(\sigma, m)$:
  - calculate $R = g^sX^c$
  - check $H(R|m) = ? c$

A multi-signature scheme [BN06]

- Space: same $G, g$
- KeyGen (by parties $i = 1, \ldots, n$):
  - Secret SignKey: $x_i \in \mathbb{Z}_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x, I}(m)$ by subset $I \subseteq \{1, \ldots, n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i = qH(X_i|R|I|m)$
  - $s = q\sum_{i \in I} s_i = \sum_{i \in I} (r_i + x_ic_i)$
  - output $\sigma = (R, s)$
- Verify$(\sigma, m)$:
  - calculate $c_i = H(X_i|R|M|I|m)$
  - check $g^s = ? R\prod_{i \in I} X_i^{c_i}$
A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

Non-threshold scheme [Sch90]
- Space: $G, g$ (group, generator)
- KeyGen (by signer):
  - Secret SignKey: $x \in \mathbb{Z}_q$
  - Public VerKey: $X = g^{-x}$
- Sign$_x(m)$ by signer:
  - $R = g^r$
  - $c = q \cdot H(R|m)$
  - $s = q \cdot r + x \cdot c$
  - output $\sigma = (s, c)$
- Verify$_x(\sigma, m)$:
  - calculate $R = g^sX^c$
  - check $H(R|m) = ? c$

A multi-signature scheme [BN06]
- Space: same $G, g$
- KeyGen (by parties $i = 1, \ldots, n$):
  - Secret SignKey: $x_i \in \mathbb{Z}_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, \ldots, n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i = q \cdot H(X_i||R||I||m)$
  - $s = q \cdot \sum_{i \in I} s_i = \sum_{i \in I} (r_i + x_ic_i)$
  - output $\sigma = (R, s)$
- Verify$(\sigma, m)$:
  - calculate $c_i = H(X_i||R||M||I||m)$
  - check $g^s = ? R \prod_{i \in I} X_i^{c_i}$

Extra slide 4/4
A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

### Non-threshold scheme [Sch90]
- **Space:** \( G, g \) (group, generator)
- **KeyGen (by signer):**
  - Secret SignKey: \( x \in \mathbb{Z}_q \)
  - Public VerKey: \( X = g^{-x} \)
- **Sign\(_x(m)\) by signer:**
  - \( R = g^r \)
  - \( c =_q H(R|m) \)
  - \( s =_q r + x \cdot c \)
  - output \( \sigma = (s, c) \)
- **Verify\(_X(\sigma, m)\):**
  - calculate \( R = g^sX^c \)
  - check \( H(R|m) =? c \)

### A multi-signature scheme [BN06]
- **Space:** same \( G, g \)
- **KeyGen (by parties \( i = 1, \ldots, n \)):**
  - Secret SignKey: \( x_i \in \mathbb{Z}_q \)
  - Public VerKey: \( X_i = g^{x_i} \)
- **Sign\(_{x,L}(m)\) by subset \( I \subseteq \{1, \ldots, n\} \):**
  - \( R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i} \)
  - \( c_i =_q H(X_i|R|I|m) \)
  - \( s =_q \sum_{i \in I} s_i = \sum_{i \in I} (r_i + x_i c_i) \)
  - output \( \sigma = (R, s) \)
- **Verify\(_(\sigma, m)\):**
  - calculate \( c_i = H(X_i|R|M|I|m) \)
  - check \( g^s =? R \prod_{i \in I} X_i^{c_i} \)
### A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

#### Non-threshold scheme [Sch90]

- **Space**: $G, g$ (group, generator)
- **KeyGen** (by signer):
  - Secret SignKey: $x \in Z_q$
  - Public VerKey: $X = g^{-x}$
- **Sign$_x(m)$** by signer:
  - $R = g^r$
  - $c = H(R || m)$
  - $s = r + x \cdot c$
  - output $\sigma = (s, c)$
- **Verify$_X(\sigma, m)$**:
  - calculate $R = g^s X^c$
  - check $H(R || m) =^? c$

#### A multi-signature scheme [BN06]

- **Space**: same $G, g$
- **KeyGen** (by parties $i = 1, \ldots, n$):
  - Secret SignKey: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- **Sign$_{x, L}(m)$** by subset $I \subseteq \{1, \ldots, n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i = H(X_i || R || I || m)$
  - $s = \sum_{i \in I} s_i = \sum_{i \in I} (r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- **Verify$(\sigma, m)$**:
  - calculate $c_i = H(X_i || R || M || I || m)$
  - check $g^s =^? R \prod_{i \in I} X_i^{c_i}$
A DL-based example: threshold Schnorr signature

*(DL = Discrete-Logarithm)*

(Next: ignore details — just making comparative remarks)

**Non-threshold scheme** [Sch90]

- **Space:** $G, g$ (group, generator)
- **KeyGen** (by signer):
  - Secret SignKey: $x \in \mathbb{Z}_q$
  - Public VerKey: $X = g^{-x}$
- **Sign$_x(m)$** by signer:
  - $R = g^r$
  - $c = H(R||m)$
  - $s = r + x \cdot c$
  - output $\sigma = (s, c)$
- **Verify$_X(\sigma, m)$**:
  - calculate $R = g^s X^c$
  - check $H(R||m) = c$

**A multi-signature scheme** [BN06]

- **Space:** same $G, g$
- **KeyGen** (by parties $i = 1, \ldots, n$):
  - Secret SignKey: $x_i \in \mathbb{Z}_q$
  - Public VerKey: $X_i = g^{x_i}$
- **Sign$_{x, L}(m)$** by subset $I \subseteq \{1, \ldots, n\}$:
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i = H(X_i||R||I||m)$
  - $s = \sum_{i \in L} s_i = \sum_{i \in I} (r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- **Verify$(\sigma, m)$**:
  - calculate $c_i = H(X_i||R||M||I||m)$
  - check $g^s = R \prod_{i \in I} X_i^{c_i}$
A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

**Non-threshold scheme [Sch90]**

- **Space**: $G, g$ (group, generator)
- **KeyGen (by signer)**:
  - **Secret SignKey**: $x \in \mathbb{Z}_q$
  - **Public VerKey**: $X = g^{-x}$
- **Sign$_x(m)$ by signer**:
  - $R = g^r$
  - $c = H(R|m)$
  - $s = r + x \cdot c$
  - output $\sigma = (s, c)$
- **Verify$_X(\sigma, m)$**:
  - calculate $R = g^sX^c$
  - check $H(R|m) = ? c$

**A multi-signature scheme [BN06]**

- **Space**: same $G, g$
- **KeyGen (by parties $i = 1, \ldots, n$)**:
  - **Secret SignKey**: $x_i \in \mathbb{Z}_q$
  - **Public VerKey**: $X_i = g^{x_i}$
- **Sign$_{x,L}(m)$ by subset $I \subseteq \{1, \ldots, n\}$**:
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i = H(X_i||R||I||m)$
  - $s = \sum_{i \in I} s_i = \sum_{i \in I} (r_i + x_ic_i)$
  - output $\sigma = (R, s)$
- **Verify($\sigma, m$)**:
  - calculate $c_i = H(X_i||R||M||I||m)$
  - check $g^s = R \prod_{i \in I} X_i^{c_i}$

Some features:
- no dealer;
- dynamic threshold (verifier decides what is acceptable);
- dynamic set of signers;
- verifying ⇒ knowing who signed.
**A DL-based example: threshold Schnorr signature**

(Next: ignore details — just making comparative remarks)

**Non-threshold scheme** [Sch90]

- Space: $G, g$ (group, generator)
- KeyGen (by signer):
  - Secret SignKey: $x \in \mathbb{Z}_q$
  - Public VerKey: $X = g^{-x}$
- $\text{Sign}_x(m)$ by signer:
  - $R = g^r$
  - $c = H(R | m)$
  - $s = r + x \cdot c$
  - output $\sigma = (s, c)$
- $\text{Verify}_X(\sigma, m)$:
  - calculate $R = g^s X^c$
  - check $H(R | m) = c$

**A multi-signature scheme** [BN06]*

- Space: same $G, g$
- KeyGen (by parties $i = 1, \ldots, n$):
  - Secret SignKey: $x_i \in \mathbb{Z}_q$
  - Public VerKey: $X_i = g^{x_i}$
- $\text{Sign}_{x_i,L}(m)$ by subset $I \subseteq \{1, \ldots, n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i = H(X_i | R | I | m)$
  - $s = \sum_{i \in L} s_i = \sum_{i \in I} (r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- $\text{Verify}(\sigma, m)$:
  - calculate $c_i = H(X_i | R | M | I | m)$
  - check $g^s = R \prod_{i \in I} X_i^{c_i}$

*Some features:

- no dealer;
- dynamic threshold (verifier decides what is acceptable);
- dynamic set of signers;
- verifying $\Rightarrow$ knowing who signed.
A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

<table>
<thead>
<tr>
<th>Non-threshold scheme</th>
<th>A multi-signature scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Space:</strong> $G, g$ (group, generator)</td>
<td><strong>Space:</strong> same $G, g$</td>
</tr>
<tr>
<td><strong>KeyGen (by signer):</strong></td>
<td><strong>KeyGen (by parties $i = 1, \ldots, n$):</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sign$_x(m)$ by signer:</strong></td>
<td><strong>Sign$_{x,L}(m)$ by subset $I \subseteq {1, \ldots, n}$</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Verify$_X(\sigma, m)$:</strong></td>
<td><strong>Verify($\sigma, m$):</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Some features: no dealer;*
A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

Non-threshold scheme [Sch90]

- Space: \( G, g \) (group, generator)
- KeyGen (by signer):
  - Secret SignKey: \( x \in Z_q \)
  - Public VerKey: \( X = g^{-x} \)
- \( \text{Sign}_x(m) \) by signer:
  - \( R = g^r \)
  - \( c = H(R|m) \)
  - \( s = r + x \cdot c \)
  - output \( \sigma = (s, c) \)
- \( \text{Verify}_X(\sigma, m) \):
  - calculate \( R = g^s X^c \)
  - check \( H(R|m) = c \)

A multi-signature scheme [BN06] *

- Space: same \( G, g \)
- KeyGen (by parties \( i = 1, \ldots , n \)):
  - Secret SignKey: \( x_i \in Z_q \)
  - Public VerKey: \( X_i = g^{x_i} \)
- \( \text{Sign}_{x,L}(m) \) by subset \( I \subseteq \{1, \ldots , n\} \)
  - \( R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i} \)
  - \( c_i = H(X_i|R|I|m) \)
  - \( s = \sum_{i \in I} s_i = \sum_{i \in I} (r_i + x_i c_i) \)
  - output \( \sigma = (R, s) \)
- \( \text{Verify}(\sigma, m) \):
  - calculate \( c_i = H(X_i|R|M|I|m) \)
  - check \( g^s = R \prod_{i \in I} X_i^{c_i} \)

*Some features: no dealer; dynamic threshold (verifier decides what is acceptable);
A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

Non-threshold scheme [Sch90]

- Space: \( G, g \) (group, generator)
- KeyGen (by signer):
  - Secret SignKey: \( x \in \mathbb{Z}_q \)
  - Public VerKey: \( X = g^{-x} \)
- \( \text{Sign}_x(m) \) by signer:
  - \( R = g^r \)
  - \( c = q \left< H(R|m) \right> \)
  - \( s = q r + x \cdot c \)
  - output \( \sigma = (s, c) \)
- \( \text{Verify}_X(\sigma, m) \):
  - calculate \( R = g^sX^c \)
  - check \( H(R|m) = ? c \)

A multi-signature scheme [BN06]*

- Space: same \( G, g \)
- KeyGen (by parties \( i = 1, \ldots, n \)):
  - Secret SignKey: \( x_i \in \mathbb{Z}_q \)
  - Public VerKey: \( X_i = g^{x_i} \)
- \( \text{Sign}_{x,I}(m) \) by subset \( I \subseteq \{1, \ldots, n\} \):
  - \( R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i} \)
  - \( c_i = q \left< H(X_i|R|I|m) \right> \)
  - \( s = q \left< \sum_{i \in I} s_i = \sum_{i \in I} (r_i + x_i c_i) \right> \)
  - output \( \sigma = (R, s) \)
- \( \text{Verify}(\sigma, m) \):
  - calculate \( c_i = H(X_i|R|M|I|m) \)
  - check \( g^s = ? R \prod_{i \in I} X_i^{c_i} \)

*Some features: no dealer; dynamic threshold (verifier decides what is acceptable); dynamic set of signers;
A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

Non-threshold scheme [Sch90]

- Space: $G, g$ (group, generator)
- KeyGen (by signer):
  - Secret SignKey: $x \in \mathbb{Z}_q$
  - Public VerKey: $X = g^{-x}$
- $\text{Sign}_x(m)$ by signer:
  - $R = g^r$
  - $c = q H(R|m)$
  - $s = q r + x \cdot c$
  - output $\sigma = (s, c)$
- $\text{Verify}_X(\sigma, m)$:
  - calculate $R = g^s X^c$
  - check $H(R|m) = ? c$

A multi-signature scheme [BN06]*

- Space: same $G, g$
- KeyGen (by parties $i = 1, \ldots, n$):
  - Secret SignKey: $x_i \in \mathbb{Z}_q$
  - Public VerKey: $X_i = g^{x_i}$
- $\text{Sign}_{x, L}(m)$ by subset $I \subseteq \{1, \ldots, n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i = q H(X_i|R|I|m)$
  - $s = q \sum_{i \in L} s_i = \sum_{i \in I} (r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- $\text{Verify}(\sigma, m)$:
  - calculate $c_i = H(X_i|R|M|I|m)$
  - check $g^s = ? R \prod_{i \in I} X_i^{c_i}$

*Some features: no dealer; dynamic threshold (verifier decides what is acceptable); dynamic set of signers; verifying $\Rightarrow$ knowing who signed.
A DL-based example: threshold Schnorr signature

(Next: ignore details — just making comparative remarks)

Non-threshold scheme [Sch90]

- Space: $G, g$ (group, generator)
- KeyGen (by signer):
  - Secret SignKey: $x \in \mathbb{Z}_q$
  - Public VerKey: $X = g^{-x}$
- Sign$_x(m)$ by signer:
  - $R = g^r$
  - $c = q \cdot H(R||m)$
  - $s = q \cdot r + x \cdot c$
  - output $\sigma = (s, c)$
- Verify$_X(\sigma, m)$:
  - calculate $R = g^sX^c$
  - check $H(R||m) = ? c$

A multi-signature scheme [BN06]*

- Space: same $G, g$
- KeyGen (by parties $i = 1, \ldots, n$):
  - Secret SignKey: $x_i \in \mathbb{Z}_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, \ldots, n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i = q \cdot H(X_i||R||I||m)$
  - $s = q \cdot \sum_{i \in I} s_i = \sum_{i \in I} (r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify$(\sigma, m)$:
  - calculate $c_i = H(X_i||R||M||I||m)$
  - check $g^s = ? R \prod_{i \in I} X_i^{c_i}$

*Some features:* no dealer; dynamic threshold (verifier decides what is acceptable); dynamic set of signers; verifying $\Rightarrow$ knowing who signed.

Extra slide 4/4