Threshold Schemes for Cryptographic Primitives

A step towards standardization?

Luís T. A. N. Brandão, Nicky Mouha, Apostol Vassilev

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

Presentation at the
NIST-CSD-CTG "Crypto Reading Club"
July 18, 2018 (Gaithersburg, MD, USA)

Minor updates in August, 2018, for online publication:
NISTIR number and link (8214); comment period;
project webpage; section 4 more succinct.

Contact email: threshold-crypto@nist.gov

Outline

1. Introduction
2. Preliminaries
3. Characterizing features
4. Some numbers
5. Steps (NISTIR, workshop)
6. Final remarks

Outline

1. Introduction

2. Preliminaries

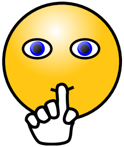
3. Characterizing features

4. Some numbers

5. Steps (NISTIR, workshop)

6. Final remarks

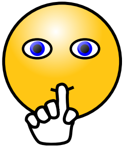
Secrets are difficult to maintain



openclipart.org/detail/76603

Secrets are difficult to maintain

A proverbial wisdom for centuries



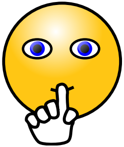
openclipart.org/detail/76603

"*Three may keep a secret, if two of them are dead.*"

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

Secrets are difficult to maintain

A proverbial wisdom for centuries



openclipart.org/detail/76603

"*Three may keep a secret, if two of them are dead.*"

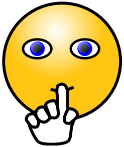
(In: *Poor Richard's Almanack.* Benjamin Franklin, 1735) [Sau34]

"*For three may kepe counseil if twain be away!*"

(In: *The Ten Commandments of Love.* Geoffrey Chaucer, 1340–1400) [Cha00]

Secrets are difficult to maintain

A proverbial wisdom for centuries



openclipart.org/detail/76603

"*Three may keep a secret, if two of them are dead.*"

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

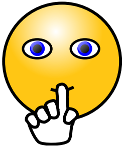
"*For three may kepe counseil if twain be away!*"

(In: *The Ten Commandments of Love*. Geoffrey Chaucer, 1340–1400) [Cha00]

Today we deal with digital information and computing

Secrets are difficult to maintain

A proverbial wisdom for centuries



openclipart.org/detail/76603

"*Three may keep a secret, if two of them are dead.*"

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

"*For three may kepe counseil if twain be away!*"

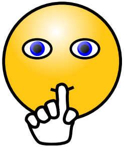
(In: *The Ten Commandments of Love*. Geoffrey Chaucer, 1340–1400) [Cha00]

Today we deal with digital information and computing

- **Cryptography** is a primary means for protecting digital information, e.g. encryption for confidentiality, signatures for integrity and authenticity, ...

Secrets are difficult to maintain

A proverbial wisdom for centuries


openclipart.org/detail/76603

"Three may keep a secret, if two of them are dead."

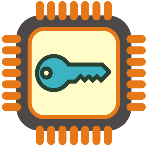
(In: *Poor Richard's Almanack.* Benjamin Franklin, 1735) [Sau34]

"For three may kepe counseil if twain be away!"

(In: *The Ten Commandments of Love.* Geoffrey Chaucer, 1340–1400) [Cha00]

Today we deal with digital information and computing

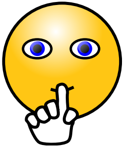
- **Cryptography** is a primary means for protecting digital information, e.g. encryption for confidentiality, signatures for integrity and authenticity, ...


openclipart.org/detail/101407

- Cryptographic effectiveness often hinges on:
  - secrecy and correctness of **cryptographic keys**

Secrets are difficult to maintain

A proverbial wisdom for centuries


openclipart.org/detail/76603

"Three may keep a secret, if two of them are dead."

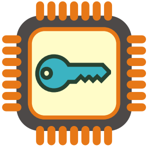
(In: *Poor Richard's Almanack.* Benjamin Franklin, 1735) [Sau34]

"For three may kepe counseil if twain be away!"

(In: *The Ten Commandments of Love.* Geoffrey Chaucer, 1340–1400) [Cha00]

Today we deal with digital information and computing

- **Cryptography** is a primary means for protecting digital information, e.g. encryption for confidentiality, signatures for integrity and authenticity, ...


openclipart.org/detail/101407

- Cryptographic effectiveness often hinges on:
  - secrecy and correctness of **cryptographic keys**
  - implementations that use **keys** in an algorithm

Crypto is affected by implementation vulnerabilities

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

Crypto is affected by implementation vulnerabilities

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

heartbleed.com

Heartbleed bug (2014)

Buffer over-read in OpenSSL's implementation of TLS reads private memory (inc. keys) from HTTPS servers [NVD14, DLK+14]

Crypto is affected by implementation vulnerabilities

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

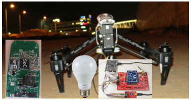
heartbleed.com

Heartbleed bug (2014)

Buffer over-read in OpenSSL's implementation of TLS reads private memory (inc. keys) from HTTPS servers [NVD14, DLK+14]

"ZigBee Chain reaction" (2017)

Side-channel attack extracts global firmware private key for Phillips Hue light-bulbs, then enabling a chain attack using the ZigBee communication protocol [RSWO17]



Crypto is affected by implementation vulnerabilities

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

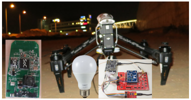
heartbleed.com

Heartbleed bug (2014)

Buffer over-read in OpenSSL's implementation of TLS reads private memory (inc. keys) from HTTPS servers [NVD14, DLK+14]

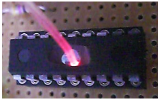
"ZigBee Chain reaction" (2017)

Side-channel attack extracts global firmware private key for Phillips Hue light-bulbs, then enabling a chain attack using the ZigBee communication protocol [RSWO17]



"Bellcore attack" (1997)

Fault injected on RSA-CRT induces incorrect signature that enables RSA factorization [BDL97]



[SH07]

Crypto is affected by implementation vulnerabilities

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

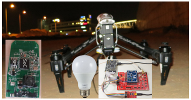
heartbleed.com

Heartbleed bug (2014)

Buffer over-read in OpenSSL's implementation of TLS reads private memory (inc. keys) from HTTPS servers [NVD14, DLK+14]

"ZigBee Chain reaction" (2017)

Side-channel attack extracts global firmware private key for Phillips Hue light-bulbs, then enabling a chain attack using the ZigBee communication protocol [RSWO17]



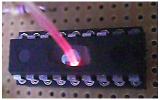
Cold-boot attacks (2009)

[Don13]

Freezing volatile memory (DRAM) enables data remanence after power-off, allowing extraction of keys from memory [HSH+09]

"Bellcore attack" (1997)

Fault injected on RSA-CRT induces incorrect signature that enables RSA factorization [BDL97]



[SH07]

Crypto is affected by implementation vulnerabilities

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

heartbleed.com

Heartbleed bug (2014)

Buffer over-read in OpenSSL's implementation of TLS reads private memory (inc. keys) from HTTPS servers [NVD14, DLK+14]

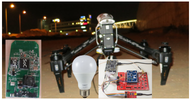
Meltdown & Spectre (2017)

Side-channel attacks reveal private memory (inc. keys) of other programs [NVD18c, LSG+18] [NVD18a, NVD18b, KGG+18]

meltdownattack.com

"ZigBee Chain reaction" (2017)

Side-channel attack extracts global firmware private key for Phillips Hue light-bulbs, then enabling a chain attack using the ZigBee communication protocol [RSWO17]



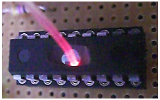
Cold-boot attacks (2009)

[Don13]

Freezing volatile memory (DRAM) enables data remanence after power-off, allowing extraction of keys from memory [HSH+09]

"Bellcore attack" (1997)

Fault injected on RSA-CRT induces incorrect signature that enables RSA factorization [BDL97]



[SH07]

Crypto is affected by implementation vulnerabilities

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

heartbleed.com

Heartbleed bug (2014)

Buffer over-read in OpenSSL's implementation of TLS reads private memory (inc. keys) from HTTPS servers [NVD14, DLK+14]

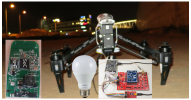
Meltdown & Spectre (2017)

Side-channel attacks reveal private memory (inc. keys) of other programs [NVD18c, LSG+18] [NVD18a, NVD18b, KGG+18]

meltdownattack.com

"ZigBee Chain reaction" (2017)

Side-channel attack extracts global firmware private key for Phillips Hue light-bulbs, then enabling a chain attack using the ZigBee communication protocol [RSWO17]



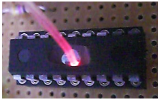
Cold-boot attacks (2009)

[Don13]

Freezing volatile memory (DRAM) enables data remanence after power-off, allowing extraction of keys from memory [HSH+09]

"Bellcore attack" (1997)

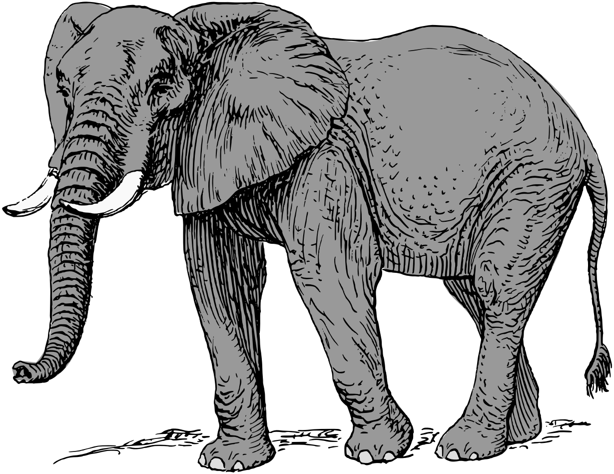
Fault injected on RSA-CRT induces incorrect signature that enables RSA factorization [BDL97]



[SH07]

It is essential to have reliable implementations of cryptographic primitives, immune to breaches in the computational environment

Single-Points of Failure!



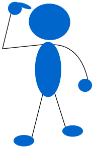
*colored-elephant.html

* = clker.com/clipart-

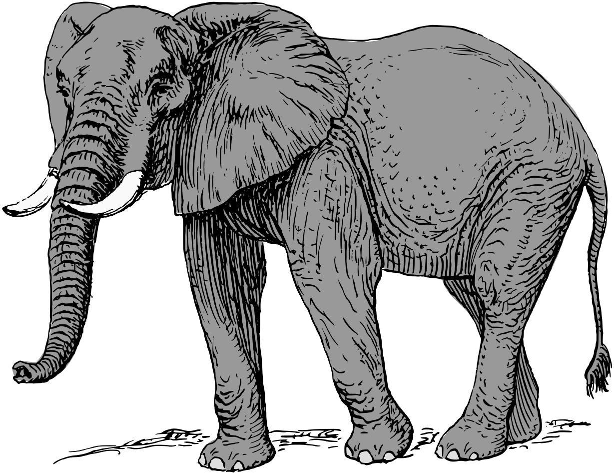
Single-Points of Failure!

Can we standardize
threshold schemes
for cryptographic
primitives
to promote the
security of crypto
implementations

*question-2.html



*4296.html

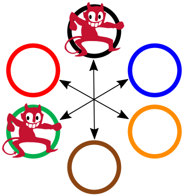


*colored-elephant.html

* = clker.com/clipart-

The threshold approach

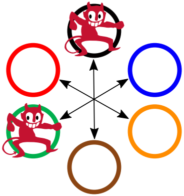
High-level idea:
Use redundancy & diversity to mitigate
the *compromise* of some (up to a threshold)
number of components (a.k.a. nodes)



The red dancing devil is from
clker.com/clipart-13643.html

The threshold approach

High-level idea:
Use redundancy & diversity to mitigate the *compromise* of some (up to a threshold) number of components (a.k.a. nodes)



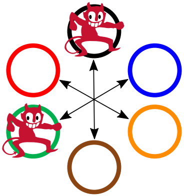
The red dancing devil is from clker.com/clipart-13643.html

The intuitive aim: improve security vs. a non-threshold scheme (depends on adversarial model)

clker.com/clipart-10778.html

The threshold approach

High-level idea:
Use redundancy & diversity to mitigate the *compromise* of some (up to a threshold) number of components (a.k.a. nodes)



The red dancing devil is from clker.com/clipart-13643.html

The intuitive aim: improve security vs. a non-threshold scheme (depends on adversarial model)

clker.com/clipart-10778.html

Note on co-existing notation:

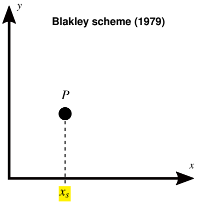
- f-out-of-n: tolerates the compromise of up to f nodes
- k-out-of-n: requires correct participation of at least k nodes

Secret Sharing Schemes (a starting point)

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

Secret Sharing Schemes (a starting point)

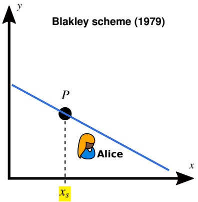
Split a secret key into n secret “shares” for storage at rest.



- The secret x_s is the x-coord of point P;

Secret Sharing Schemes (a starting point)

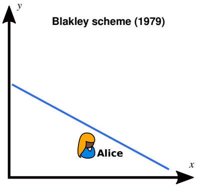
Split a secret key into n secret “shares” for storage at rest.



- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;

Secret Sharing Schemes (a starting point)

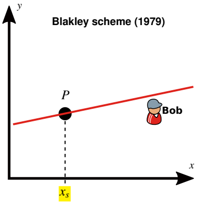
Split a secret key into n secret "shares" for storage at rest.



- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;

Secret Sharing Schemes (a starting point)

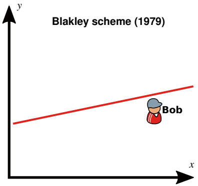
Split a secret key into n secret "shares" for storage at rest.



- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;

Secret Sharing Schemes (a starting point)

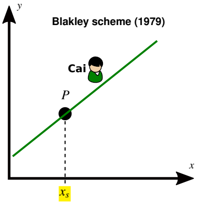
Split a secret key into n secret "shares" for storage at rest.



- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;

Secret Sharing Schemes (a starting point)

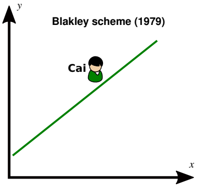
Split a secret key into n secret “shares” for storage at rest.



- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;

Secret Sharing Schemes (a starting point)

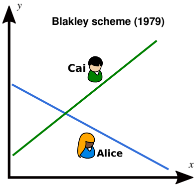
Split a secret key into n secret "shares" for storage at rest.



- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;

Secret Sharing Schemes (a starting point)

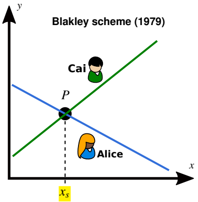
Split a secret key into n secret “shares” for storage at rest.



- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

Secret Sharing Schemes (a starting point)

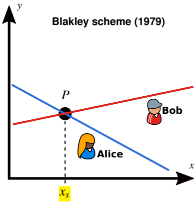
Split a secret key into n secret “shares” for storage at rest.



- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

Secret Sharing Schemes (a starting point)

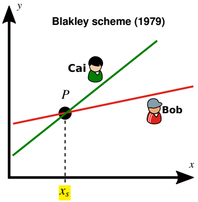
Split a secret key into n secret "shares" for storage at rest.



- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

Secret Sharing Schemes (a starting point)

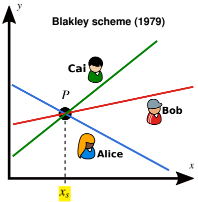
Split a secret key into n secret "shares" for storage at rest.



- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

Secret Sharing Schemes (a starting point)

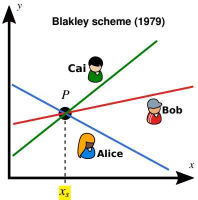
Split a secret key into n secret "shares" for storage at rest.

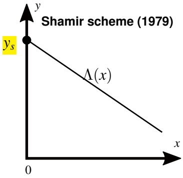


- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

Secret Sharing Schemes (a starting point)

Split a secret key into n secret "shares" for storage at rest.



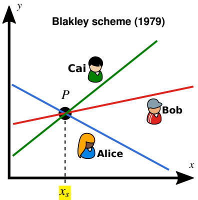


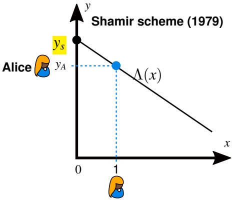
- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

- The secret y_s is the y-coord of $\Lambda(x = 0)$;

Secret Sharing Schemes (a starting point)

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



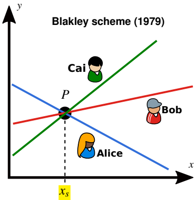


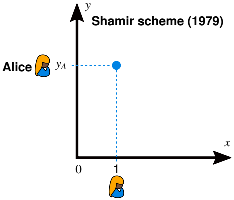
- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

- The secret y_s is the y-coord of $\Lambda(x=0)$;
- Each share is a point $(\Lambda(i), i)$ of line Λ;

Secret Sharing Schemes (a starting point)

Split a secret key into n secret "shares" for storage at rest.



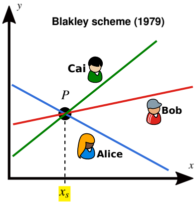


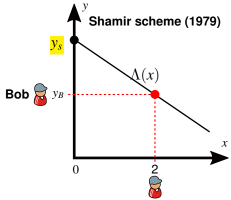
- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

- The secret y_s is the y-coord of $\Lambda(x=0)$;
- Each share is a point $(\Lambda(i), i)$ of line Λ;
- A single point does not convey info of y_s;

Secret Sharing Schemes (a starting point)

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



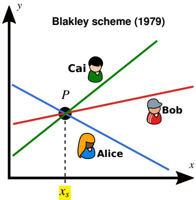


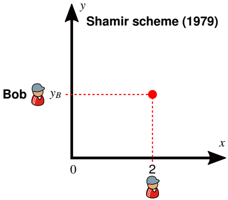
- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

- The secret y_s is the y-coord of $\Lambda(x = 0)$;
- Each share is a point $(\Lambda(i), i)$ of line Λ;
- A single point does not convey info of y_s;

Secret Sharing Schemes (a starting point)

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



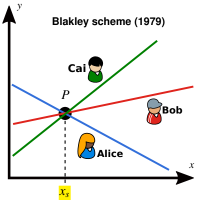


- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

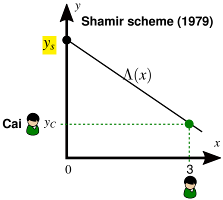
- The secret y_s is the y-coord of $\Lambda(x = 0)$;
- Each share is a point $(\Lambda(i), i)$ of line Λ;
- A single point does not convey info of y_s;

Secret Sharing Schemes (a starting point)

Split a secret key into n secret "shares" for storage at rest.



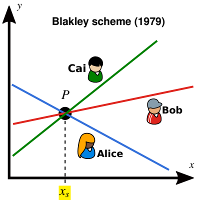
- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

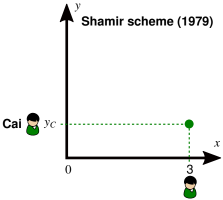


- The secret y_s is the y-coord of $\Lambda(x=0)$;
- Each share is a point $(\Lambda(i), i)$ of line Λ;
- A single point does not convey info of y_s;

Secret Sharing Schemes (a starting point)

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



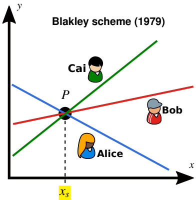


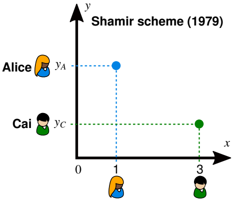
- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

- The secret y_s is the y-coord of $\Lambda(x=0)$;
- Each share is a point $(\Lambda(i), i)$ of line Λ;
- A single point does not convey info of y_s;

Secret Sharing Schemes (a starting point)

Split a secret key into n secret "shares" for storage at rest.



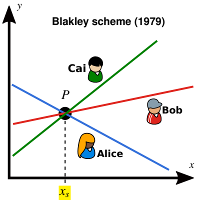


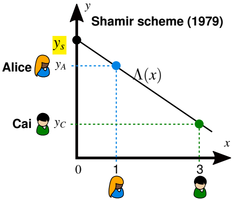
- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

- The secret y_s is the y-coord of $\Lambda(x=0)$;
- Each share is a point $(\Lambda(i), i)$ of line Λ;
- A single point does not convey info of y_s;
- Two distinct **points** (shares) reveal Λ.

Secret Sharing Schemes (a starting point)

Split a secret key into n secret "shares" for storage at rest.



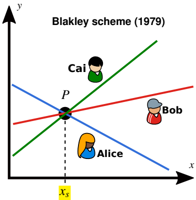


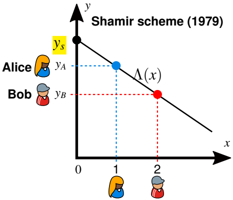
- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

- The secret y_s is the y-coord of $\Lambda(x = 0)$;
- Each share is a point $(\Lambda(i), i)$ of line Λ;
- A single point does not convey info of y_s;
- Two distinct **points** (shares) reveal Λ.

Secret Sharing Schemes (a starting point)

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



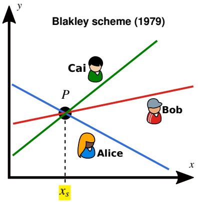


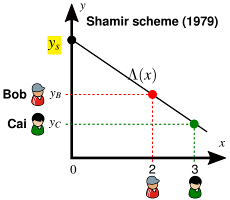
- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

- The secret y_s is the y-coord of $\Lambda(x = 0)$;
- Each share is a point $(\Lambda(i), i)$ of line Λ;
- A single point does not convey info of y_s;
- Two distinct **points** (shares) reveal Λ.

Secret Sharing Schemes (a starting point)

Split a secret key into n secret "shares" for storage at rest.



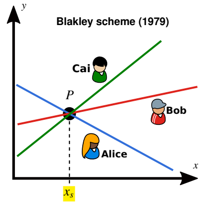


- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

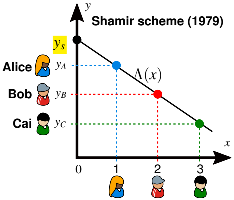
- The secret y_s is the y-coord of $\Lambda(x = 0)$;
- Each share is a point $(\Lambda(i), i)$ of line Λ;
- A single point does not convey info of y_s;
- Two distinct **points** (shares) reveal Λ.

Secret Sharing Schemes (a starting point)

Split a secret key into n secret "shares" for storage at rest.



Humanoid cliparts:
clker.com/clipart-*.html
Alice: *=2478
Bob: *=2482
Cai: *=2479



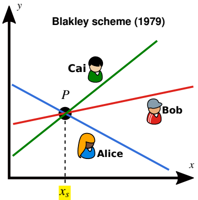
- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

- The secret y_s is the y-coord of $\Lambda(x = 0)$;
- Each share is a point $(\Lambda(i), i)$ of line Λ;
- A single point does not convey info of y_s;
- Two distinct **points** (shares) reveal Λ.

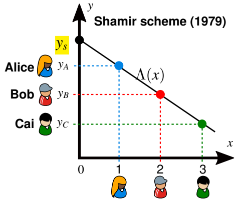
Each share in isolation has no information about the secret.
k shares (in the example, $k = 2$) are enough to recover the secret.

Secret Sharing Schemes (a starting point)

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



Humanoid cliparts:
clker.com/clipart-*.html
Alice: *=2478
Bob: *=2482
Cai: *=2479



- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

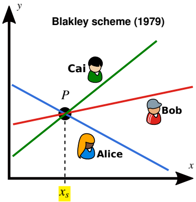
- The secret y_s is the y-coord of $\Lambda(x = 0)$;
- Each share is a point $(\Lambda(i), i)$ of line Λ;
- A single point does not convey info of y_s;
- Two distinct **points** (shares) reveal Λ.

Each share in isolation has no information about the secret.
k shares (in the example, $k = 2$) are enough to recover the secret.

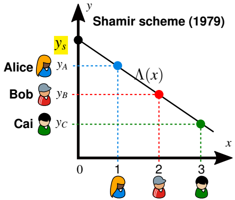
But, how to avoid recombining the key when the key is needed by an algorithm?

Secret Sharing Schemes (a starting point)

Split a secret key into n secret "shares" for storage at rest.



Humanoid cliparts:
clker.com/clipart-*.html
Alice: *=2478
Bob: *=2482
Cai: *=2479



- The secret x_s is the x-coord of point P;
- Each share is a distinct line crossing P;
- A single line does not convey info of x_s;
- Two distinct **lines** (shares) reveal P.

- The secret y_s is the y-coord of $\Lambda(x = 0)$;
- Each share is a point $(\Lambda(i), i)$ of line Λ;
- A single point does not convey info of y_s;
- Two distinct **points** (shares) reveal Λ.

Each share in isolation has no information about the secret.
k shares (in the example, $k = 2$) are enough to recover the secret.

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

Use threshold schemes for cryptographic primitives (next)

Our current step

Image adapted from:
openclipart.org/detail/283392

Devise initial questions for discussion
towards standardization and validation of
threshold schemes for cryptographic primitives.

Our current step

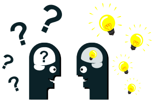


Image adapted from:
openclipart.org/detail/283392

Devise initial questions for discussion
towards standardization and validation of
threshold schemes for cryptographic primitives.

Goals for this presentation:

Our current step

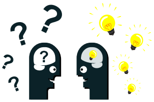


Image adapted from:
openclipart.org/detail/283392

Devise initial questions for discussion
towards standardization and validation of
threshold schemes for cryptographic primitives.

Goals for this presentation:

- Convey high-dimensionality of the problem

Our current step

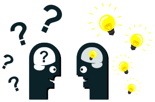


Image adapted from:
openclipart.org/detail/283392

Devise initial questions for discussion towards standardization and validation of threshold schemes for cryptographic primitives.

Goals for this presentation:

- Convey high-dimensionality of the problem
- Convey a few technical examples (not delving into much detail)

Our current step

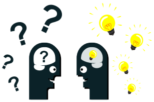


Image adapted from: openclipart.org/detail/283392

Devise initial questions for discussion towards standardization and validation of threshold schemes for cryptographic primitives.

Goals for this presentation:

- Convey high-dimensionality of the problem
- Convey a few technical examples (not delving into much detail)
- Motivate feedback (NISTIR draft) and engagement (next steps...)

Our current step

Image adapted from:
openclipart.org/detail/283392

Devise initial questions for discussion towards standardization and validation of threshold schemes for cryptographic primitives.

Goals for this presentation:

- Convey high-dimensionality of the problem
- Convey a few technical examples (not delving into much detail)
- Motivate feedback (NISTIR draft) and engagement (next steps...)
- Suggest moving forward (with challenges)

Outline

1. Introduction

2. Preliminaries

3. Characterizing features

4. Some numbers

5. Steps (NISTIR, workshop)

6. Final remarks

Example: RSA signature (or decryption)

Example: RSA signature (or decryption)

Classical scheme [RSA78]

- KeyGen (by signer):
  - Public Modulus: $N = p \cdot q$
  - Secret SignKey: d
  - Public VerKey: e (with $d \cdot e =_{\phi} 1$)
- Sign(m): $\sigma =_N m^d$
- Verify(σ, m): $\sigma^e \stackrel{?}{=}_N m$

Example: RSA signature (or decryption)

Classical scheme [RSA78]

- KeyGen (by signer):
  - Public Modulus: $N = p \cdot q$
  - Secret SignKey: d
  - Public VerKey: e (with $d \cdot e =_\phi 1$)
- Sign(m): $\sigma =_N m^d$
- Verify(σ, m): $\sigma^e \stackrel{?}{=}_N m$

A 3-out-of-3 threshold scheme

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

Example: RSA signature (or decryption)

Classical scheme [RSA78]

- KeyGen (by signer):
  - Public Modulus: $N = p \cdot q$
  - Secret SignKey: d
  - Public VerKey: e (with $d \cdot e =_\phi 1$)
- Sign(m): $\sigma =_N m^d$
- Verify(σ, m): $\sigma^e \stackrel{?}{=}_N m$

A 3-out-of-3 threshold scheme

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

Example: RSA signature (or decryption)

Classical scheme [RSA78]

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

A 3-out-of-3 threshold scheme

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

Example: RSA signature (or decryption)

Classical scheme [RSA78]

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

A 3-out-of-3 threshold scheme

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

About the threshold scheme:

SignKey d not recombined;

Example: RSA signature (or decryption)

Classical scheme [RSA78]

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

A 3-out-of-3 threshold scheme

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

About the threshold scheme:

SignKey d not recombined; can *reshare* d leaving e fixed;

Example: RSA signature (or decryption)

Classical scheme [RSA78]

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

A 3-out-of-3 threshold scheme

- KeyGen (by dealer):
  - Same N, d, e
  - SubKeys: d_1, d_2, d_3: $d_1 + d_2 + d_3 =_\phi d$
- Sign'(m): { separate: $s_i =_N m^{d_i} : i = 1, 2, 3$

  combine: $\sigma =_N s_1 \cdot s_2 \cdot s_3$ }
- Verify(σ, m): $\sigma^e \overset{?}{=}_N m$

About the threshold scheme:

SignKey d not recombined; can *reshare* d leaving e fixed; same σ;

Example: RSA signature (or decryption)

Classical scheme [RSA78]

- KeyGen (by signer):
  - Public Modulus: $N = p \cdot q$
  - Secret SignKey: d
  - Public VerKey: e (with $d \cdot e =_\phi 1$)
- Sign(m): $\sigma =_N m^d$
- Verify(σ, m): $\sigma^e \stackrel{?}{=}_N m$

A 3-out-of-3 threshold scheme

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

About the threshold scheme:

SignKey d not recombined; can *reshare* d leaving e fixed; same σ; efficient!

Example: RSA signature (or decryption)

Classical scheme [RSA78]

- KeyGen (by signer):
  - Public Modulus: $N = p \cdot q$
  - Secret SignKey: d
  - Public VerKey: e (with $d \cdot e =_\phi 1$)
- Sign(m): $\sigma =_N m^d$
- Verify(σ, m): $\sigma^e \stackrel{?}{=}_N m$

A 3-out-of-3 threshold scheme

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

About the threshold scheme:

SignKey d not recombined; can *reshare* d leaving e fixed; same σ; efficient!

Facilitating setting: $\exists$ dealer;

Example: RSA signature (or decryption)

Classical scheme [RSA78]

- KeyGen (by signer):
  - Public Modulus: $N = p \cdot q$
  - Secret SignKey: d
  - Public VerKey: e (with $d \cdot e =_{\phi} 1$)
- Sign(m): $\sigma =_N m^d$
- Verify(σ, m): $\sigma^e \stackrel{?}{=}_N m$

A 3-out-of-3 threshold scheme

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

About the threshold scheme:

SignKey d not recombined; can *reshare* d leaving e fixed; same σ; efficient!

Facilitating setting: $\exists$ dealer; $\exists$ homomorphism;

Example: RSA signature (or decryption)

Classical scheme [RSA78]

- KeyGen (by signer):
  - Public Modulus: $N = p \cdot q$
  - Secret SignKey: d
  - Public VerKey: e (with $d \cdot e =_\phi 1$)
- Sign(m): $\sigma =_N m^d$
- Verify(σ, m): $\sigma^e \stackrel{?}{=}_N m$

A 3-out-of-3 threshold scheme

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

About the threshold scheme:

SignKey d not recombined; can *reshare* d leaving e fixed; same σ; efficient!

Facilitating setting: $\exists$ dealer; $\exists$ homomorphism; all parties know m.

Example: RSA signature (or decryption)

Classical scheme [RSA78]

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

A 3-out-of-3 threshold scheme

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

About the threshold scheme:

SignKey d not recombined; can *reshare* d leaving e fixed; same σ; efficient!

Facilitating setting: $\exists$ dealer; $\exists$ homomorphism; all parties know m.

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

Example: RSA signature (or decryption)

Classical scheme [RSA78]

- KeyGen (by signer):
  - Public Modulus: $N = p \cdot q$
  - Secret SignKey: d
  - Public VerKey: e (with $d \cdot e =_{\phi} 1$)
- Sign(m): $\sigma =_N m^d$
- Verify(σ, m): $\sigma^e \stackrel{?}{=}_N m$

A 3-out-of-3 threshold scheme

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

About the threshold scheme:

SignKey d not recombined; can *reshare* d leaving e fixed; same σ; efficient!

Facilitating setting: $\exists$ dealer; $\exists$ homomorphism; all parties know m.

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

- Can it be improved to withstand f malicious signers**?**

Example: RSA signature (or decryption)

Classical scheme [RSA78]

- KeyGen (by signer):
  - Public Modulus: $N = p \cdot q$
  - Secret SignKey: d
  - Public VerKey: e (with $d \cdot e =_\phi 1$)
- Sign(m): $\sigma =_N m^d$
- Verify(σ, m): $\sigma^e \stackrel{?}{=}_N m$

A 3-out-of-3 threshold scheme

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

About the threshold scheme:

SignKey d not recombined; can *reshare* d leaving e fixed; same σ; efficient!

Facilitating setting: $\exists$ dealer; $\exists$ homomorphism; all parties know m.

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

- Can it be improved to withstand f malicious signers**?** **Yes** (hint in next slide)

Example: RSA signature (or decryption)

Classical scheme [RSA78]

- KeyGen (by signer):
  - Public Modulus: $N = p \cdot q$
  - Secret SignKey: d
  - Public VerKey: e (with $d \cdot e =_\phi 1$)
- Sign(m): $\sigma =_N m^d$
- Verify(σ, m): $\sigma^e \stackrel{?}{=}_N m$

A 3-out-of-3 threshold scheme

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

About the threshold scheme:

SignKey d not recombined; can *reshare* d leaving e fixed; same σ; efficient!

Facilitating setting: $\exists$ dealer; $\exists$ homomorphism; all parties know m.

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

- Can it be improved to withstand f malicious signers**? Yes** (hint in next slide)
- Can it be done: $\nexists$ dealer, $\nexists$ homomorphisms, secret-shared m**?**

Example: RSA signature (or decryption)

Classical scheme [RSA78]

- KeyGen (by signer):
  - Public Modulus: $N = p \cdot q$
  - Secret SignKey: d
  - Public VerKey: e (with $d \cdot e =_\phi 1$)
- Sign(m): $\sigma =_N m^d$
- Verify(σ, m): $\sigma^e \stackrel{?}{=}_N m$

A 3-out-of-3 threshold scheme

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

About the threshold scheme:

SignKey d not recombined; can *reshare* d leaving e fixed; same σ; efficient!

Facilitating setting: $\exists$ dealer; $\exists$ homomorphism; all parties know m.

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

- Can it be improved to withstand f malicious signers**? Yes** (hint in next slide)
- Can it be done: $\nexists$ dealer, $\nexists$ homomorphisms, secret-shared m**? Yes** (... later)

Example: Robust k-out-of-n Threshold RSA Signature

Example: Robust k-out-of-n Threshold RSA Signature

Can the previous scheme be enhanced? E.g., to work:

Example: Robust k-out-of-n Threshold RSA Signature

Can the previous scheme be enhanced? E.g., to work:

- if only a subset I with k' parties is available, with $k \leq k' \leq n$?

Example: Robust k-out-of-n Threshold RSA Signature

Can the previous scheme be enhanced? E.g., to work:

- if only a subset I with k' parties is available, with $k \leq k' \leq n$?
- robust against malicious parties (i.e., incorrect signature-shares)?

Example: Robust k-out-of-n Threshold RSA Signature

Can the previous scheme be enhanced? E.g., to work:

- if only a subset I with k' parties is available, with $k \leq k' \leq n$?
- robust against malicious parties (i.e., incorrect signature-shares)?

Yes! (e.g., [Sho00])

Example: Robust k-out-of-n Threshold RSA Signature

Can the previous scheme be enhanced? E.g., to work:

- if only a subset I with k' parties is available, with $k \leq k' \leq n$?
- robust against malicious parties (i.e., incorrect signature-shares)?

Yes! (e.g., [Sho00]) At high level:

Example: Robust k-out-of-n Threshold RSA Signature

Can the previous scheme be enhanced? E.g., to work:

- if only a subset I with k' parties is available, with $k \leq k' \leq n$?
- robust against malicious parties (i.e., incorrect signature-shares)?

Yes! (e.g., [Sho00]) At high level:

- **RSA homomorphism** $\rightarrow$ can combine (slightly tweaked) sub-signatures

Example: Robust k-out-of-n Threshold RSA Signature

Can the previous scheme be enhanced? E.g., to work:

- if only a subset I with k' parties is available, with $k \leq k' \leq n$?
- robust against malicious parties (i.e., incorrect signature-shares)?

Yes! (e.g., [Sho00]) At high level:

- **RSA homomorphism** $\rightarrow$ can combine (slightly tweaked) sub-signatures (via polynomial interpolation in the exponent — get $m^{f(0)}$ from $m^{\lambda'_{i,I} f(i)} : i \in I$);

Example: Robust k-out-of-n Threshold RSA Signature

Can the previous scheme be enhanced? E.g., to work:

- if only a subset I with k' parties is available, with $k \leq k' \leq n$?
- robust against malicious parties (i.e., incorrect signature-shares)?

Yes! (e.g., [Sho00]) At high level:

- **RSA homomorphism** $\rightarrow$ can combine (slightly tweaked) sub-signatures (via polynomial interpolation in the exponent — get $m^{f(0)}$ from $m^{\lambda'_{i,I} f(i)}$: $i \in I$);

Example: Robust k-out-of-n Threshold RSA Signature

Can the previous scheme be enhanced? E.g., to work:

- if only a subset I with k' parties is available, with $k \leq k' \leq n$?
- robust against malicious parties (i.e., incorrect signature-shares)?

Yes! (e.g., [Sho00]) At high level:

- **RSA homomorphism** $\rightarrow$ can combine (slightly tweaked) sub-signatures (via polynomial interpolation in the exponent — get $m^{f(0)}$ from $m^{\lambda'_{i,I} f(i)} : i \in I$);
- Dealer **commits** (one-time) to every share $\rightarrow$ each sub-signer gives a **NIZKP** (non-interactive zero-knowledge proofs) of correct sub-signature.

Example: Robust k-out-of-n Threshold RSA Signature

Can the previous scheme be enhanced? E.g., to work:

- if only a subset I with k' parties is available, with $k \leq k' \leq n$?
- robust against malicious parties (i.e., incorrect signature-shares)?

Yes! (e.g., [Sho00]) At high level:

- **RSA homomorphism** $\rightarrow$ can combine (slightly tweaked) sub-signatures (via polynomial interpolation in the exponent — get $m^{f(0)}$ from $m^{\lambda'_{i,I}f(i)} : i \in I$);
- Dealer **commits** (one-time) to every share $\rightarrow$ each sub-signer gives a **NIZKP** (non-interactive zero-knowledge proofs) of correct sub-signature.

It is efficient:

- Size: final signature is as original
- Sub-signer computation: original, plus produce 1 NIZKP (2 exps)
- Combiner computation: original, plus 1 ext-GCD and verify NIZKPs ($2 \cdot k$ exps)

A DL-based example: threshold Schnorr signature

(DL = Discrete-Logarithm)

A DL-based example: threshold Schnorr signature

(DL = Discrete-Logarithm)

(Next clicks: ignore details — just making comparative remarks)

A DL-based example: threshold Schnorr signature

(DL = Discrete-Logarithm)

(Next clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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 $I \subseteq \{1, \ldots, n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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 clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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 clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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 clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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 clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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 clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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 $I \subseteq \{1, \ldots, n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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 clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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 clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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 clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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 $I \subseteq \{1, \ldots, n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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 clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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 $I \subseteq \{1, \ldots, n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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 clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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 clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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 clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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:**

A DL-based example: threshold Schnorr signature

(DL = Discrete-Logarithm)

(Next clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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 $i = 1, ..., n$):
  - Secret SignKey: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- $\text{Sign}_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - 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;

A DL-based example: threshold Schnorr signature

(DL = Discrete-Logarithm)

(Next clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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

(DL = Discrete-Logarithm)

(Next clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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

(DL = Discrete-Logarithm)

(Next clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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 clicks: 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:
  - $R = g^r$
  - $c =_q H(R||m)$
  - $s =_q r + x \cdot c$
  - 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: $x_i \in Z_q$
  - Public VerKey: $X_i = g^{x_i}$
- Sign$_{x,L}(m)$ by subset $I \subseteq \{1, ..., n\}$
  - $R = \prod_{i \in I} R_i = \prod_{i \in I} g^{r_i}$
  - $c_i =_q H(X_i||R||I||m)$
  - $s =_q \sum_{i \in L} s_i = \sum_{i \in I}(r_i + x_i c_i)$
  - output $\sigma = (R, s)$
- Verify(σ, 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.

Comparing thresholds

3-out-of-3 decryption:

clker.com/clipart-encryption.html

2-out-of-3 signature:

clker.com/clipart-3712.html

Comparing thresholds

3-out-of-3 decryption:

clker.com/clipart-encryption.html

- 3 nodes needed to decrypt (availability: $k = 3, f = 0$);

2-out-of-3 signature:

clker.com/clipart-3712.html

Comparing thresholds

3-out-of-3 decryption:

clker.com/clipart-encryption.html

- 3 nodes needed to decrypt (availability: $k = 3, f = 0$);
- key is secret if at least 1 node does not leak its key share (secrecy of key: $k = 1, f = 2$).

2-out-of-3 signature:

clker.com/clipart-3712.html

Comparing thresholds

3-out-of-3 decryption:

clker.com/clipart-encryption.html

- 3 nodes needed to decrypt (availability: $k = 3, f = 0$);
- key is secret if at least 1 node does not leak its key share (secrecy of key: $k = 1, f = 2$).

2-out-of-3 signature:

clker.com/clipart-3712.html

- 2 nodes needed to sign (availability: $k = 2, f = 1$);

Comparing thresholds

3-out-of-3 decryption:

clker.com/clipart-encryption.html

- 3 nodes needed to decrypt (availability: $k = 3, f = 0$);
- key is secret if at least 1 node does not leak its key share (secrecy of key: $k = 1, f = 2$).

2-out-of-3 signature:

clker.com/clipart-3712.html

- 2 nodes needed to sign (availability: $k = 2, f = 1$);
- key secret if at least 2 nodes do not leak their key shares (secrecy of key: $k = 2, f = 1$).

Comparing thresholds

3-out-of-3 decryption:

clker.com/clipart-encryption.html

- 3 nodes needed to decrypt (availability: $k = 3, f = 0$);
- key is secret if at least 1 node does not leak its key share (secrecy of key: $k = 1, f = 2$).

2-out-of-3 signature:

clker.com/clipart-3712.html

- 2 nodes needed to sign (availability: $k = 2, f = 1$);
- key secret if at least 2 nodes do not leak their key shares (secrecy of key: $k = 2, f = 1$).

Do these provide better security than a non-threshold scheme ($n = 1, f = 0$)?

Comparing thresholds

3-out-of-3 decryption:

clker.com/clipart-encryption.html

- 3 nodes needed to decrypt (availability: $k = 3, f = 0$);
- key is secret if at least 1 node does not leak its key share (secrecy of key: $k = 1, f = 2$).

2-out-of-3 signature:

clker.com/clipart-3712.html

- 2 nodes needed to sign (availability: $k = 2, f = 1$);
- key secret if at least 2 nodes do not leak their key shares (secrecy of key: $k = 2, f = 1$).

Do these provide better security than a non-threshold scheme ($n = 1, f = 0$)?

- Are there common failure modes? (e.g., is breaking 1 node as difficult as breaking 3?)

Comparing thresholds

3-out-of-3 decryption:

clker.com/clipart-encryption.html

- 3 nodes needed to decrypt (availability: $k = 3, f = 0$);
- key is secret if at least 1 node does not leak its key share (secrecy of key: $k = 1, f = 2$).

2-out-of-3 signature:

clker.com/clipart-3712.html

- 2 nodes needed to sign (availability: $k = 2, f = 1$);
- key secret if at least 2 nodes do not leak their key shares (secrecy of key: $k = 2, f = 1$).

Do these provide better security than a non-threshold scheme $(n = 1, f = 0)$?

- Are there common failure modes? (e.g., is breaking 1 node as difficult as breaking 3?)
- Even if independent failure mode: can breaking 2 out of 3 be easier than 1 out of 1?

Comparing thresholds

3-out-of-3 decryption:

clker.com/clipart-encryption.html

- 3 nodes needed to decrypt (availability: $k = 3, f = 0$);
- key is secret if at least 1 node does not leak its key share (secrecy of key: $k = 1, f = 2$).

2-out-of-3 signature:

clker.com/clipart-3712.html

- 2 nodes needed to sign (availability: $k = 2, f = 1$);
- key secret if at least 2 nodes do not leak their key shares (secrecy of key: $k = 2, f = 1$).

Do these provide better security than a non-threshold scheme ($n = 1, f = 0$)?

- Are there common failure modes? (e.g., is breaking 1 node as difficult as breaking 3?)
- Even if independent failure mode: can breaking 2 out of 3 be easier than 1 out of 1?
- Is plaintext secrecy affected? (does the client send/receive it whole or shared?)

Comparing thresholds

3-out-of-3 decryption:

clker.com/clipart-encryption.html

- 3 nodes needed to decrypt (availability: $k = 3, f = 0$);
- key is secret if at least 1 node does not leak its key share (secrecy of key: $k = 1, f = 2$).

2-out-of-3 signature:

clker.com/clipart-3712.html

- 2 nodes needed to sign (availability: $k = 2, f = 1$);
- key secret if at least 2 nodes do not leak their key shares (secrecy of key: $k = 2, f = 1$).

Do these provide better security than a non-threshold scheme ($n = 1, f = 0$)?

- Are there common failure modes? (e.g., is breaking 1 node as difficult as breaking 3?)
- Even if independent failure mode: can breaking 2 out of 3 be easier than 1 out of 1?
- Is plaintext secrecy affected? (does the client send/receive it whole or shared?)
- May the implementation bring new security problems?

Comparing thresholds

3-out-of-3 decryption:

clker.com/clipart-encryption.html

- 3 nodes needed to decrypt (availability: $k = 3, f = 0$);
- key is secret if at least 1 node does <u>not</u> leak its key share (secrecy of key: $k = 1, f = 2$).

2-out-of-3 signature:

clker.com/clipart-3712.html

- 2 nodes needed to sign (availability: $k = 2, f = 1$);
- key secret if at least 2 nodes do <u>not</u> leak their key shares (secrecy of key: $k = 2, f = 1$).

Do these provide better security than a non-threshold scheme ($n = 1, f = 0$)?

- Are there common failure modes? (e.g., is breaking 1 node as difficult as breaking 3?)
- Even if independent failure mode: can breaking 2 out of 3 be easier than 1 out of 1?
- Is plaintext secrecy affected? (does the client send/receive it whole or shared?)
- May the implementation bring new security problems?

> "k-out-of-n" or "f-out-of-n" is not a sufficient characterization for a comprehensive security assertion

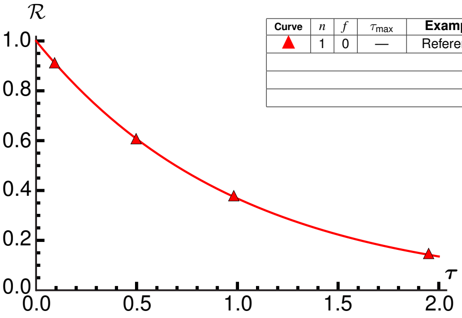
Reliability — one metric of security

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

Reliability — one metric of security

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

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



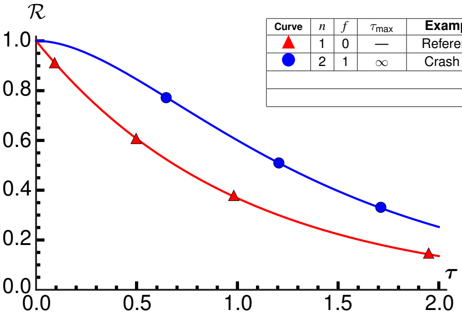
Curve	n	f	$\tau_{\max}$	Example
▲	1	0	—	Reference

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

Reliability — one metric of security

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

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



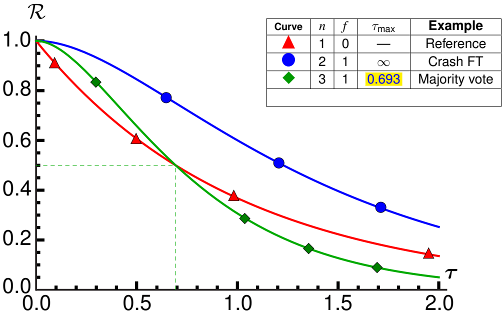
Curve	n	f	$\tau_{\max}$	Example
▲	1	0	—	Reference
●	2	1	∞	Crash FT

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

Reliability — one metric of security

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

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

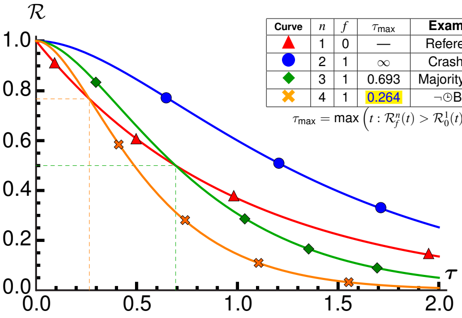


[BB12] Time normalized: $\tau = 1$ is the *expected time to failure* (ETTF) of a node

Reliability — one metric of security

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

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



Curve	n	f	$\tau_{\max}$	Example
▲	1	0	—	Reference
●	2	1	∞	Crash FT
◆	3	1	0.693	Majority vote
✖	4	1	0.264	¬◷BFT

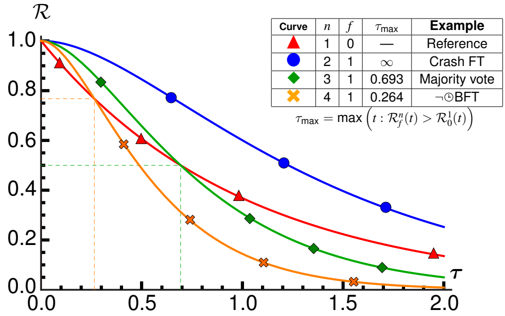
$\tau_{\max} = \max\left(t : \mathcal{R}_f^n(t) > \mathcal{R}_0^1(t)\right)$

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

Reliability — one metric of security

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

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



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

Reliability can be degraded when increasing the threshold (f), even if nodes fail independently

Improve reliability with rejuvenations

Rejuvenation (recovery of nodes): compromised state $\rightarrow$ healthy state

Improve reliability with rejuvenations

Rejuvenation (recovery of nodes): compromised state $\rightarrow$ healthy state

- Examples: replace device, patch vulnerability, update or reset a state, ...

Improve reliability with rejuvenations

Rejuvenation (recovery of nodes): compromised state $\rightarrow$ healthy state

- Examples: replace device, patch vulnerability, update or reset a state, ...
- Attenuates the reliability-degradation for long mission time (MT)

Improve reliability with rejuvenations

Rejuvenation (recovery of nodes): compromised state $\rightarrow$ healthy state

- Examples: replace device, patch vulnerability, update or reset a state, ...
- Attenuates the reliability-degradation for long mission time (MT)
- Increases availability (another metric: % of secure time), even for ∞ MT

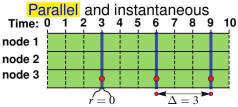
Improve reliability with rejuvenations

Rejuvenation (recovery of nodes): compromised state → healthy state

- Examples: replace device, patch vulnerability, update or reset a state, ...
- Attenuates the reliability-degradation for long mission time (MT)
- Increases availability (another metric: % of secure time), even for ∞ MT

Rejuvenation modes:

- parallel



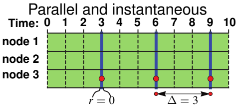
Improve reliability with rejuvenations

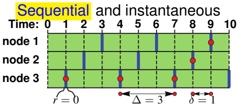
Rejuvenation (recovery of nodes): compromised state → healthy state

- Examples: replace device, patch vulnerability, update or reset a state, ...
- Attenuates the reliability-degradation for long mission time (MT)
- Increases availability (another metric: % of secure time), even for ∞ MT

Rejuvenation modes:

- parallel vs. sequential





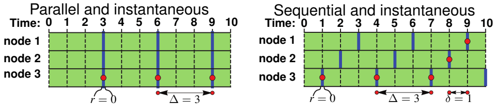
Improve reliability with rejuvenations

Rejuvenation (recovery of nodes): compromised state → healthy state

- Examples: replace device, patch vulnerability, update or reset a state, ...
- Attenuates the reliability-degradation for long mission time (MT)
- Increases availability (another metric: % of secure time), even for ∞ MT

Rejuvenation modes:

- parallel vs. sequential
- offline



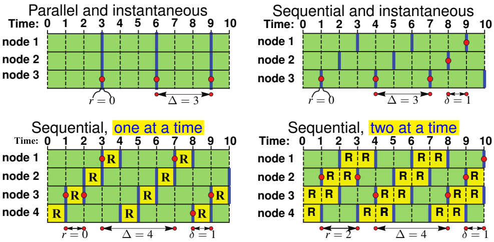
Improve reliability with rejuvenations

Rejuvenation (recovery of nodes): compromised state $\rightarrow$ healthy state

- Examples: replace device, patch vulnerability, update or reset a state, ...
- Attenuates the reliability-degradation for long mission time (MT)
- Increases availability (another metric: % of secure time), even for ∞ MT

Rejuvenation modes:

- parallel vs. sequential
- offline vs. online



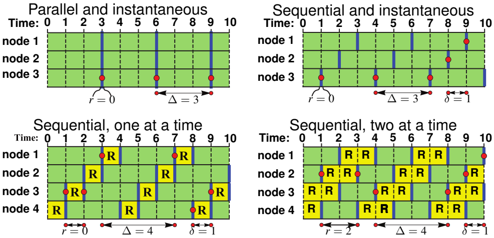
Improve reliability with rejuvenations

Rejuvenation (recovery of nodes): compromised state → healthy state

- Examples: replace device, patch vulnerability, update or reset a state, ...
- Attenuates the reliability-degradation for long mission time (MT)
- Increases availability (another metric: % of secure time), even for ∞ MT

Rejuvenation modes:

- parallel vs. sequential
- offline vs. online
- reactive vs. proactive (detected vs. stealth intrusions; frequency?)



[BB12]

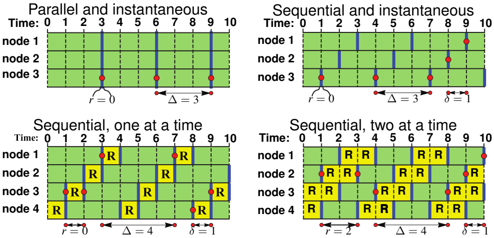
Improve reliability with rejuvenations

Rejuvenation (recovery of nodes): compromised state → healthy state

- Examples: replace device, patch vulnerability, update or reset a state, ...
- Attenuates the reliability-degradation for long mission time (MT)
- Increases availability (another metric: % of secure time), even for ∞ MT

Rejuvenation modes:

- parallel vs. sequential
- offline vs. online
- reactive vs. proactive (detected vs. stealth intrusions; frequency?)



[BB12]

Other effects:

- adds cost, implementation complexity ... (new vulnerabilities?)

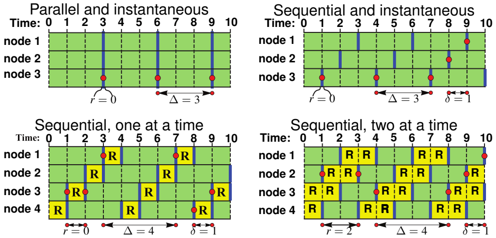
Improve reliability with rejuvenations

Rejuvenation (recovery of nodes): compromised state → healthy state

- Examples: replace device, patch vulnerability, update or reset a state, ...
- Attenuates the reliability-degradation for long mission time (MT)
- Increases availability (another metric: % of secure time), even for ∞ MT

Rejuvenation modes:

- parallel vs. sequential
- offline vs. online
- reactive vs. proactive (detected vs. stealth intrusions; frequency?)



[BB12]

Other effects:

- adds cost, implementation complexity ... (new vulnerabilities?)
- parallel rejuvenation may imply period of unavailable service

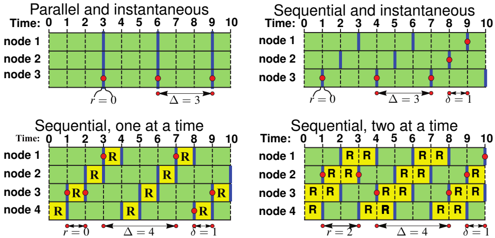
Improve reliability with rejuvenations

Rejuvenation (recovery of nodes): compromised state $\rightarrow$ healthy state

- Examples: replace device, patch vulnerability, update or reset a state, ...
- Attenuates the reliability-degradation for long mission time (MT)
- Increases availability (another metric: % of secure time), even for ∞ MT

Rejuvenation modes:

- parallel vs. sequential
- offline vs. online
- reactive vs. proactive (detected vs. stealth intrusions; frequency?)



[BB12]

Other effects:

- adds cost, implementation complexity ... (new vulnerabilities?)
- parallel rejuvenation may imply period of unavailable service
- sequential rejuvenations may still allow a mobile attacker to persist

Another model

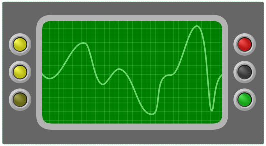
What if all nodes are compromised (e.g., leaky) from the start?

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)



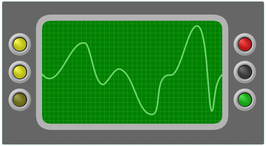
openclipart.org/detail/172330

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)



openclipart.org/detail/172330

Challenge questions:

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

Outline

1. Introduction

2. Preliminaries

3. Characterizing features

4. Some numbers

5. Steps (NISTIR, workshop)

6. Final remarks

What kind of threshold scheme?

What kind of threshold scheme?

To reflect on a threshold scheme, let us characterize the system.

What kind of threshold scheme?

To reflect on a threshold scheme, let us characterize the system.

Four main features:

1. Kinds of threshold
2. Communication interfaces
3. Executing platform
4. Setup and maintenance

What kind of threshold scheme?

To reflect on a threshold scheme, let us characterize the system.

Four main features:

1. Kinds of threshold
2. Communication interfaces
3. Executing platform
4. Setup and maintenance

Each feature contains distinct options that affect security in a different way.

What kind of threshold scheme?

To reflect on a threshold scheme, let us characterize the system.

Four main features:

1. Kinds of threshold
2. Communication interfaces
3. Executing platform
4. Setup and maintenance

Each feature contains distinct options that affect security in a different way.

A characterization provides a better context for security assertions.

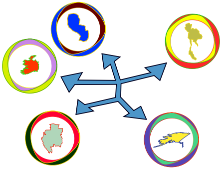
1. Kinds of threshold

1. Kinds of threshold

- Need k-out-of-n good nodes (or tolerate up to f-out-of-n bad nodes) for which values k and f? for which security properties?

1. Kinds of threshold

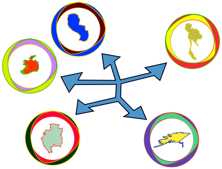
- Need k-out-of-n good nodes (or tolerate up to f-out-of-n bad nodes) for which values k and f? for which security properties?
  (some pairs (n, f) may not be possible, e.g., some settings require $n \geq 3f + 1$)



openclipart.org/detail/71491

1. Kinds of threshold

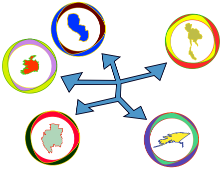
- Need k-out-of-n good nodes (or tolerate up to f-out-of-n bad nodes) for which values k and f? for which security properties?
  (some pairs (n, f) may not be possible, e.g., some settings require $n \geq 3f + 1$)
- Levels of *diversity* (e.g., location, software, shares) vs. non-diversity across the n components (common vulnerabilities)?



openclipart.org/detail/71491

1. Kinds of threshold

- Need k-out-of-n good nodes (or tolerate up to f-out-of-n bad nodes) for which values k and f? for which security properties?
  (some pairs (n, f) may not be possible, e.g., some settings require $n \geq 3f + 1$)
- Levels of *diversity* (e.g., location, software, shares) vs. non-diversity across the n components (common vulnerabilities)?
- Variable threshold and number of nodes?
  (changing parameters may need its own protocol)



openclipart.org/detail/71491

2. Communication interfaces

2. Communication interfaces

- Inter-node: structure (e.g., star vs. clique)? channel protection?

openclipart.org/detail/190624

2. Communication interfaces

- Inter-node: structure (e.g., star vs. clique)? channel protection?
- Client $\leftrightarrow$ crypto module: proxy? primary node? shares?

openclipart.org/detail/190624

2. Communication interfaces

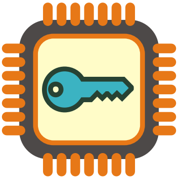
- Inter-node: structure (e.g., star vs. clique)? channel protection?
- Client $\leftrightarrow$ crypto module: proxy? primary node? shares?
- Is client unaware vs. needs proof of threshold computation?

openclipart.org/detail/190624

3. Executing platform

3. Executing platform

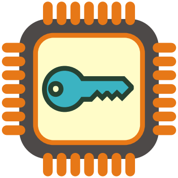
- Single (multi-chip) device vs. multi-party (e.g., multiple computers)



openclipart.org/detail/101407

3. Executing platform

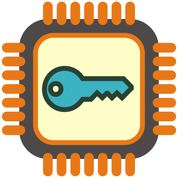
- Single (multi-chip) device vs. multi-party (e.g., multiple computers)
- Software vs. hardware



openclipart.org/detail/101407

3. Executing platform

- Single (multi-chip) device vs. multi-party (e.g., multiple computers)
- Software vs. hardware
- Additional trusted machinery? (global clock, proxy, RNG, combiner)

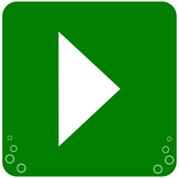


openclipart.org/detail/101407

4. Setup and maintenance

4. Setup and maintenance

- How to bootstrap?
  - dealer vs. SMPC-initialization of secret shares
  - crypto setup assumption: identities, PKI, synchrony, ...?

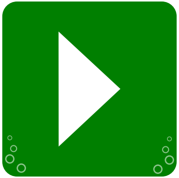


openclipart.org/detail/161401

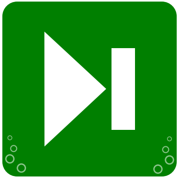
4. Setup and maintenance

- How to bootstrap?
  - dealer vs. SMPC-initialization of secret shares
  - crypto setup assumption: identities, PKI, synchrony, ...?

- Rejuvenation modes: (parallel vs. sequential, online vs. offline, ...)



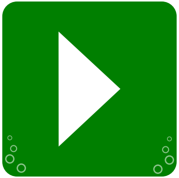
openclipart.org/detail/161401



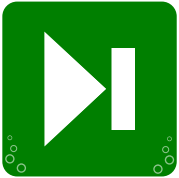
openclipart.org/detail/161389

4. Setup and maintenance

- How to bootstrap?
  - dealer vs. SMPC-initialization of secret shares
  - crypto setup assumption: identities, PKI, synchrony, ...?
- Rejuvenation modes: (parallel vs. sequential, online vs. offline, ...)
- Diversity: offline pre-computation vs. on-the-fly vs. limited set



openclipart.org/detail/161401



openclipart.org/detail/161389

Deployment context

Deployment context

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

Deployment context

- **Application context.** Should it affect security requirements?
  - If app layer verifies signature correctness, is it okay to use non-robust signature scheme?
  - Encryption more difficult?

openclipart.org/detail/281637

Deployment context

- **Application context.** Should it affect security requirements?
  - If app layer verifies signature correctness, is it okay to use non-robust signature scheme?
  - Encryption more difficult?

openclipart.org/detail/281637

- **Conceivable attack types.**

clker.com/clipart-10778

  - 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?
  - If app layer verifies signature correctness, is it okay to use non-robust signature scheme?
  - Encryption more difficult?

openclipart.org/detail/281637

- **Conceivable attack types.**

clker.com/clipart-10778

  - 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

Outline

1. Introduction

2. Preliminaries

3. Characterizing features

4. Some numbers

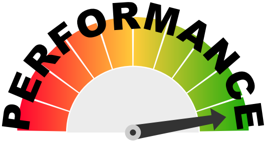
5. Steps (NISTIR, workshop)

6. Final remarks

Some numbers

[Content adapted in online version]

Just for an intuition:
brief notes on
recent efficiency claims.

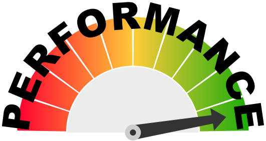


openclipart.org/detail/291407

Some numbers

[Content adapted in online version]

Just for an intuition:
brief notes on
recent efficiency claims.



openclipart.org/detail/291407

Recent research works significantly improve concrete efficiency of threshold schemes for cryptographic primitives, e.g.:

Some numbers

[Content adapted in online version]

Just for an intuition:
brief notes on
recent efficiency claims.

openclipart.org/detail/291407

Recent research works significantly improve concrete efficiency of threshold schemes for cryptographic primitives, e.g.:

- threshold signatures and threshold key generation
- threshold AES evaluation (SMPC-based)
- threshold circuit design of symmetric primitives
- threshold random-number generation (coin-tossing)

Outline

1. Introduction

2. Preliminaries

3. Characterizing features

4. Some numbers

5. Steps (NISTIR, workshop)

6. Final remarks

NIST Internal Report

NIST Internal Report

We've been preparing a Draft NISTIR (8214), reflecting towards the possibility of standards for threshold schemes.

NIST Internal Report

We've been preparing a Draft NISTIR (8214), reflecting towards the possibility of standards for threshold schemes. It intends to:

- Position a set of relevant questions (mostly unanswered)
- Layout the need to describe characterizing features
- Motivate development of a criteria for selection of schemes
- Be a reference to motivate engagement from stakeholders

NIST Internal Report

We've been preparing a Draft NISTIR (8214), reflecting towards the possibility of standards for threshold schemes. It intends to:

- Position a set of relevant questions (mostly unanswered)
- Layout the need to describe characterizing features
- Motivate development of a criteria for selection of schemes
- Be a reference to motivate engagement from stakeholders

Any feedback you provide is welcome and valuable!

NIST Internal Report

We've been preparing a Draft NISTIR (8214), reflecting towards the possibility of standards for threshold schemes. It intends to:

- Position a set of relevant questions (mostly unanswered)
- Layout the need to describe characterizing features
- Motivate development of a criteria for selection of schemes
- Be a reference to motivate engagement from stakeholders

Any feedback you provide is welcome and valuable!

Timeline:

- Post public draft still in July.
- First round of public comments till October 22, 2018.

NIST Internal Report

We've been preparing a Draft NISTIR (8214), reflecting towards the possibility of standards for threshold schemes. It intends to:

- Position a set of relevant questions (mostly unanswered)
- Layout the need to describe characterizing features
- Motivate development of a criteria for selection of schemes
- Be a reference to motivate engagement from stakeholders

Any feedback you provide is welcome and valuable!

Timeline:

- Post public draft still in July.
- First round of public comments till October 22, 2018.

Next slides: some representative questions on flexibility and validation challenges

Flexibility of features and parameters

Standard → implementation validation → deployment

Flexibility of features and parameters

Standard $\rightarrow$ implementation validation $\rightarrow$ deployment

- **Standard.** What flexibility of features & parameters should a threshold-scheme standard allow?
- **Validation.** What should be delimited at validation phase (e.g., validated only for $n \geq 2f + 1$; particular hardware; shares initialized with SMPC, ...)
- **Deployment.** What remains flexible for deployment? (e.g., f; how to (re-)initialize shares? dealer vs. SMPC?)

clker.com/clipart-stretching-navy.html

Flexibility of features and parameters

Standard $\rightarrow$ implementation validation $\rightarrow$ deployment

- **Standard.** What flexibility of features & parameters should a threshold-scheme standard allow?
- **Validation.** What should be delimited at validation phase (e.g., validated only for $n \geq 2f + 1$; particular hardware; shares initialized with SMPC, ...)
- **Deployment.** What remains flexible for deployment? (e.g., f; how to (re-)initialize shares? dealer vs. SMPC?)

clker.com/clipart-stretching-navy.html

- What should be tested/validated vs. can rely on vendor assertion?
  - E.g., how to ensure that good randomness will be used?
  - E.g., how to validate rejuvenations (schedule, diversity, ...)?

Flexibility of features and parameters

Standard $\rightarrow$ implementation validation $\rightarrow$ deployment

- **Standard.** What flexibility of features & parameters should a threshold-scheme standard allow?
- **Validation.** What should be delimited at validation phase (e.g., validated only for $n \geq 2f + 1$; particular hardware; shares initialized with SMPC, ...)
- **Deployment.** What remains flexible for deployment? (e.g., f; how to (re-)initialize shares? dealer vs. SMPC?)

clker.com/clipart-stretching-navy.html

- What should be tested/validated vs. can rely on vendor assertion?
  - E.g., how to ensure that good randomness will be used?
  - E.g., how to validate rejuvenations (schedule, diversity, ...)?

Answers may to a certain extent depend on what can be assessed by test & validation procedures (some of which to develop)!

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:

- When using crypto, federal agencies can only use **standardized** algorithms and **validated** implementations [tC96]
- FIPS 140-2 defines, for cryptographic modules, 4 security levels: subsets of applicable security assertions [NIS01]

The validation challenge

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

Validation is needed:

- When using crypto, federal agencies can only use **standardized** algorithms and **validated** implementations [tC96]
- FIPS 140-2 defines, for cryptographic modules, 4 security levels: subsets of applicable security assertions [NIS01]

Automation of validation in the CAVP and CMVP:

- Automate CAVP by Fall 2018, based on newly developed ACVP [NIS18]
- Ongoing pilots (Google, Red Hat) on automated module validations

Legend: ACVP (Automated Cryptographic Validation Protocol) CAVP (Cryptographic Algorithm Validation Program); CMVP (Cryptographic Module Validation Program); FIPS (Federal Information Processing Standards).

Some open questions about validation

Some open questions about validation

- **Security assertions:** what should be validated about a threshold scheme implementation?

clker.com/clipart-25196.html

Some open questions about validation

- **Security assertions:** what should be validated about a threshold scheme implementation?
- **Checklist of attacks:** should a validation level (= set of security assertions) contain a checklist of attack scenarios and security properties?

clker.com/clipart-25196.html

Some open questions about validation

- **Security assertions:** what should be validated about a threshold scheme implementation?
- **Checklist of attacks:** should a validation level (= set of security assertions) contain a checklist of attack scenarios and security properties?


clker.com/clipart-25196.html

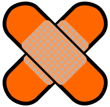
- **Adaptation:** how should validation procedures and assertions vary with (or adapt to) threshold features and application context?
  - with/without dealer, executing platform, rejuvenation modes, ...

Modularity

Modularity

Patch-and-revalidate scenario. If a f-out-of-n (for availability) system has *diversity* of implementation across nodes, then:

- a new vulnerability in a node can be patched offline
- a node can be audited / upgraded / revalidated offline

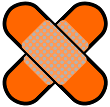


openclipart.org/detail/22712

Modularity

Patch-and-revalidate scenario. If a f-out-of-n (for availability) system has *diversity* of implementation across nodes, then:

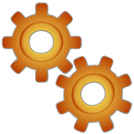
- a new vulnerability in a node can be patched offline
- a node can be audited / upgraded / revalidated offline



openclipart.org/detail/22712

Base primitives. Is it useful to standardize/define certain modules? (composability argument)

- secret sharing
- commitments
- ZK proofs
- oblivious transfer
- ... (other SMPC tools)



clker.com/clipart-2948.html

A Workshop?

We want to find answers in collaboration with stakeholders!

A Workshop?

We want to find answers in collaboration with stakeholders!

Can we do it in an open workshop?

- learn the state-of-the-art and survey the area
- define a criteria for a call proposals for threshold schemes
- tentative month: March 2019?

Outline

1. Introduction

2. Preliminaries

3. Characterizing features

4. Some numbers

5. Steps (NISTIR, workshop)

6. Final remarks

Summary

Summary

- Crypto implementations will have vulnerabilities!
- Threshold schemes have potential to avoid single-points of failure.
- There are long standing solutions ... there are also recent ones

Summary

- Crypto implementations will have vulnerabilities!
- Threshold schemes have potential to avoid single-points of failure.
- There are long standing solutions ... there are also recent ones

To evaluate threshold schemes, we should characterize:

- Features (thresholds, interfaces, platform, setup and maintenance)
- Adversarial model: goals, capabilities, vectors
- Different effects (improve vs. degrade) on diverse security properties
- New complexity from threshold approach? (bugs, efficiency, ...)

Summary

- Crypto implementations will have vulnerabilities!
- Threshold schemes have potential to avoid single-points of failure.
- There are long standing solutions ... there are also recent ones

To evaluate threshold schemes, we should characterize:

- Features (thresholds, interfaces, platform, setup and maintenance)
- Adversarial model: goals, capabilities, vectors
- Different effects (improve vs. degrade) on diverse security properties
- New complexity from threshold approach? (bugs, efficiency, ...)

Standardizing a threshold scheme would also entail:

- Deciding what remains flexible up to validation and/or deployment phases
- Develop test procedures and security assertions for validation

Moving forward

The end goals:

- standardize threshold schemes for cryptographic primitives
- develop guidelines for validation
- promote good practices of deployment

Moving forward

The end goals:

- standardize threshold schemes for cryptographic primitives
- develop guidelines for validation
- promote good practices of deployment

Meanwhile:

- We would appreciate feedback on the Draft NISTIR (8214).
- We would like to extend an open invitation for you to participate in upcoming steps.

Thanks

Thank you for your attention!

Threshold Schemes for Cryptographic Primitives
A step towards standardization?

Contact us at threshold-crypto@nist.gov

Check updates https://csrc.nist.gov/Projects/Threshold-Cryptography

References I

[BB12] L. T. A. N. Brandão and A. N. Bessani. *On the reliability and availability of replicated and rejuvenating systems under stealth attacks and intrusions*. Journal of the Brazilian Computer Society, 18(1):61–80, 2012. DOI:10.1007/s13173-012-0062-x.

[BDL97] D. Boneh, R. A. DeMillo, and R. J. Lipton. *On the Importance of Checking Cryptographic Protocols for Faults*. In W. Fumy (ed.), Advances in Cryptology — EUROCRYPT '97, pages 37–51, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. DOI:10.1007/3-540-69053-0˙4.

[BN06] M. Bellare and G. Neven. *Multi-signatures in the Plain public-Key Model and a General Forking Lemma*. In Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS '06, pages 390–399, New York, NY, USA, 2006. ACM. DOI:10.1145/1180405.1180453.

[Cha00] G. Chaucer. *The Ten Commandments of Love*, 1340–1400. See "For three may kepe counseil if twain be away!" in the "Secretnesse" stanza of the poem. https://sites.fas.harvard.edu/ chaucer/special/lifemann/love/ten-comm.html. Accessed: July 2018.

[DLK+14] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman. *The Matter of Heartbleed*. In Proceedings of the 2014 Conference on Internet Measurement Conference, IMC '14, pages 475–488, New York, NY, USA, 2014. ACM. DOI:10.1145/2663716.2663755.

[Don13] D. Donzai. *Using Cold Boot Attacks and Other Forensic Techniques in Penetration Tests*, 2013. https://www.ethicalhacker.net/features/root/using-cold-boot-attacks-forensic-techniques-penetration-tests/. Accessed: July 2018.

[HSH+09] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. *Lest We Remember: Cold-boot Attacks on Encryption Keys*. Commun. ACM, 52(5):91–98, May 2009. DOI:10.1145/1506409.1506429.

[KGG+18] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. *Spectre Attacks: Exploiting Speculative Execution*. ArXiv e-prints, January 2018. arXiv:1801.01203.

[LSG+18] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg. *Meltdown*. ArXiv e-prints, jan 2018. arXiv:1801.01207.

[NIS01] NIST. *Security Requirements for Cryptographic Modules, Federal Information Processing Standard (FIPS) 140-2*, 2001. DOI:10.6028/NIST.FIPS.140-2.

[NIS18] NIST. *Automated Cryptographic Validation Protocol*. https://github.com/usnistgov/ACVP, 2018.

[NVD14] NVD. *National Vulnerability Database — CVE-2014-0160*. https://nvd.nist.gov/vuln/detail/CVE-2014-0160, 2014.

[NVD18a] NVD. *National Vulnerability Database — CVE-2017-5715*. https://nvd.nist.gov/vuln/detail/CVE-2017-5715, 2018.

[NVD18b] NVD. *National Vulnerability Database — CVE-2017-5753*. https://nvd.nist.gov/vuln/detail/CVE-2017-5753, 2018.

[NVD18c] NVD. *National Vulnerability Database — CVE-2017-5754*. https://nvd.nist.gov/vuln/detail/CVE-2017-5754, 2018.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. *A method for obtaining digital signatures and public-key cryptosystems*. Communications of the ACM, 21(2):120–126, 1978. DOI:10.1145/359340.359342.

[RSWO17] E. Ronen., A. Shamir, A.-O. Weingarten, and C. O'Flynn. *IoT Goes Nuclear: Creating a ZigBee Chain Reaction*. IEEE Symposium on Security and Privacy, pages 195–212, 2017. DOI:10.1109/SP.2017.14.

[Sau34] R. Saunders. *Poor Richard's Almanack — 1735*. Benjamin Franklin, 1734.

[Sch90] C. P. Schnorr. *Efficient Identification and Signatures for Smart Cards*. In G. Brassard (ed.), Advances in Cryptology — CRYPTO' 89 Proceedings, pages 239–252, New York, NY, 1990. Springer New York. DOI:10.1007/0-387-34805-0˙22.

[SH07] J.-M. Schmidt and M. Hutter. *Optical and EM Fault-Attacks on CRT-based RSA: Concrete Results*, pages 61–67. Verlag der Technischen Universität Graz, 2007.

[Sho00] V. Shoup. *Practical Threshold Signatures*. In B. Preneel (ed.), Advances in Cryptology — EUROCRYPT 2000, pages 207–220, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. DOI:10.1007/3-540-45539-6˙15.

[tC96] U. S. 104th Congress. *Information Technology Management Reform Act. Public Law 104–106, Section 5131*, 1996. https://www.dol.gov/ocfo/media/regs/ITMRA.pdf.