BLS signatures, hashing to curves, and more: Dispatches from the IETF

Riad S. Wahby, Dan Boneh

Stanford

August 18th, 2019

People

- BLS signatures authors:

  Sergey Gorbunov, Hoeteck Wee, Zhenfei Zhang

- Hash-to-curve authors:

  Armando Faz-Hernández, Sam Scott, Nick Sullivan, Chris Wood

- Folks whose feedback has been crucial:

  Björn Haase, Dan Harkins, Leo Reyzin,

  Michael Scott, Shoko Yonezawa

Why standardize? +philosophy$^{\epsilon}$

The obvious one: interoperability

Why standardize? +philosophy$^\epsilon$

The obvious one: interoperability

But also: efficiency, security

Why standardize? +philosophy$^\epsilon$

The obvious one: interoperability

Invalid curve attack [BMM00]

Mallory		**Bob**
P of order ℓ (on the *wrong curve*!)		secret $\beta \in \mathbb{Z}_q$
	P →	
		$Q = P^\beta$
	← Q	
Extract β mod ℓ		

Why standardize? +philosophy$^\epsilon$

The obvious one: interoperability

But also: efficiency, security

Practice "polite crypto" [EWD1300]

➔ Do the careful thinking up front
so that your users don't have to!

Why standardize? +philosophy$^{\epsilon}$

The obvious one: interoperability

But also: efficiency, security

Practice "polite crypto" [EWD1300]

➔ Do the careful thinking up front
so that your users don't have to!

A forcing function for pragmatism

➔ Users will ignore bad or confusing standards...
...so make choices (but only the good ones)

1. Standardizing advanced crypto with the IETF

2. BLS signatures, hash-to-curve, and more

Standardizing advanced crypto with the IETF: the CFRG

Internet Engineering Task Force

"We believe in rough consensus and running code."

Standardizing advanced crypto with the IETF: the CFRG

Internet Engineering Task Force

“We believe in rough consensus and running code.”

Crypto Forum Research Group

“serves as a bridge between theory and practice, bringing new cryptographic techniques to the Internet community”

Standardizing advanced crypto with the IETF: the CFRG

Internet Engineering Task Force

"We believe in rough consensus and running code."

Crypto Forum Research Group

"serves as a bridge between theory and practice, bringing new cryptographic techniques to the Internet community"

CFRG publishes "Informational" RFCs

➔ can be incorporated by "Standards Track" RFCs, e.g., TLS 1.3 incorporates the curve25519 RFC

Standardizing advanced crypto with the IETF: the CFRG

Internet Engineering Task Force

"We believe in rough consensus and running code."

Crypto Forum Research Group

"serves as a bridge between theory and practice, bringing new cryptographic techniques to the Internet community"

CFRG publishes "Informational" RFCs

➔ can be incorporated by "Standards Track" RFCs, e.g., TLS 1.3 incorporates the curve25519 RFC

CFRG has an active mailing list, too!

➔ `https://irtf.org/cfrg`

CFRG standardization process—theory

Per [RFC5743]:

1. CFRG prepares a new “internet draft”
2. IRTF reviews it
3. IESG reviews it
4. RFC Editor prepares and publishes it

CFRG standardization process—theory

Per [RFC5743]:

1. CFRG prepares a new “internet draft”

   technical vetting—correctness

2. IRTF reviews it

3. IESG reviews it

4. RFC Editor prepares and publishes it

CFRG standardization process—theory

Per [RFC5743]:

1. CFRG prepares a new “internet draft”
   technical vetting—correctness
2. IRTF reviews it
   editorial vetting—clarity
3. IESG reviews it
4. RFC Editor prepares and publishes it

CFRG standardization process—theory

Per [RFC5743]:

1. CFRG prepares a new “internet draft”
   technical vetting—correctness
2. IRTF reviews it
   editorial vetting—clarity
3. IESG reviews it
   “political” vetting—is CFRG the right group?
4. RFC Editor prepares and publishes it

CFRG standardization process—theory

Per [RFC5743]:

1. CFRG prepares a new "internet draft"
   technical vetting—correctness
2. IRTF reviews it
   editorial vetting—clarity
3. IESG reviews it
   "political" vetting—is CFRG the right group?
4. RFC Editor prepares and publishes it
   fine-toothed combing

CFRG standardization process—practice

1. Build consensus: the world needs this protocol
   - stakeholders: the community at large (plus CFRG)

CFRG standardization process—practice

1. Build consensus: the world needs this protocol
   - stakeholders: the community at large (plus CFRG)
2. Write an "individual draft"
   - solicit feedback from stakeholders
   - `https://github.com/ietf-gitwg/using-github`

CFRG standardization process—practice

1. Build consensus: the world needs this protocol
   - stakeholders: the community at large (plus CFRG)
2. Write an “individual draft”
   - solicit feedback from stakeholders
   - `https://github.com/ietf-gitwg/using-github`
3. CFRG call for adoption
   - vote on CFRG mailing list: should this group work on this document? who will read and give feedback?

CFRG standardization process—practice

1. Build consensus: the world needs this protocol
   - stakeholders: the community at large (plus CFRG)
2. Write an “individual draft”
   - solicit feedback from stakeholders
   - `https://github.com/ietf-gitwg/using-github`
3. CFRG call for adoption
   - vote on CFRG mailing list: should this group work on this document? who will read and give feedback?
4. edit, implement, present updates at IETF meetings

CFRG standardization process—practice

1. Build consensus: the world needs this protocol
   - stakeholders: the community at large (plus CFRG)
2. Write an "individual draft"
   - solicit feedback from stakeholders
   - `https://github.com/ietf-gitwg/using-github`
3. CFRG call for adoption
   - vote on CFRG mailing list: should this group work on this document? who will read and give feedback?
4. edit, implement, present updates at IETF meetings
5. CFRG last call (for objections, comments, etc.)

CFRG standardization process—practice

1. Build consensus: the world needs this protocol
   - stakeholders: the community at large (plus CFRG)
2. Write an "individual draft"
   - solicit feedback from stakeholders
   - `https://github.com/ietf-gitwg/using-github`
3. CFRG call for adoption
   - vote on CFRG mailing list: should this group work on this document? who will read and give feedback?
4. edit, implement, present updates at IETF meetings
5. CFRG last call (for objections, comments, etc.)
6. hand off document to IRTF, etc.

CFRG standardization process—how long does it take?

Examples (from `https://datatracker.ietf.org`):

- curve25519/curve448 [RFC7748]: about 1 year

  12 drafts in total
  IRTF, IESG reviews took a few days each
  RFC Editor queue took 3 months

CFRG standardization process—how long does it take?

Examples (from `https://datatracker.ietf.org`):

- curve25519/curve448 [RFC7748]: about 1 year
  - 12 drafts in total
  - IRTF, IESG reviews took a few days each
  - RFC Editor queue took 3 months
- BLS signatures (WIP): 6 months so far
  - 2 drafts so far
- Hash-to-curve (WIP): 17 months so far
  - 5 drafts so far

What about patents?

There's an RFC for that! [RFC8179]

- If you own a patent, you *must* disclose it.
- If you know of a patent, you *should* disclose it.

What about patents?

There's an RFC for that! [RFC8179]

If you own a patent, you *must* disclose it.

If you know of a patent, you *should* disclose it.

➔ IETF will ask rights holders for written assurance that patents will be licensed to implementors.

What about patents?

There's an RFC for that! [RFC8179]

If you own a patent, you *must* disclose it.

If you know of a patent, you *should* disclose it.

➔ IETF will ask rights holders for written assurance that patents will be licensed to implementors.

IETF Security Area won't specify "must-implement" protocols that have royalty encumbrances.

What about patents?

There's an RFC for that! [RFC8179]

- If you own a patent, you *must* disclose it.
- If you know of a patent, you *should* disclose it.
- ➔ IETF will ask rights holders for written assurance that patents will be licensed to implementors.

IETF Security Area won't specify "must-implement" protocols that have royalty encumbrances.

- Royalty-free "RAND-z" licenses are OK; commitments not to assert patents are better; unencumbered technologies are best.

What about patents?

There's an RFC for that! [RFC8179]

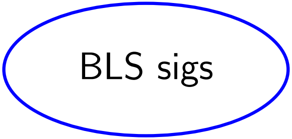
- If you own a patent, you *must* disclose it.
- If you know of a patent, you *should* disclose it.
- ➔ IETF will ask rights holders for written assurance that patents will be licensed to implementors.

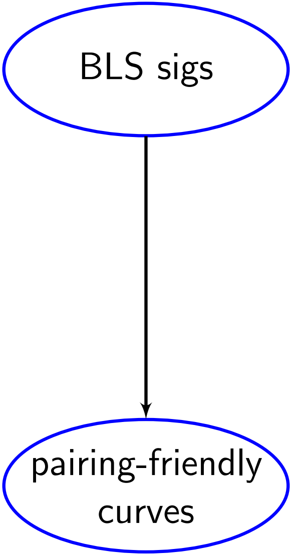
IETF Security Area won't specify "must-implement" protocols that have royalty encumbrances.

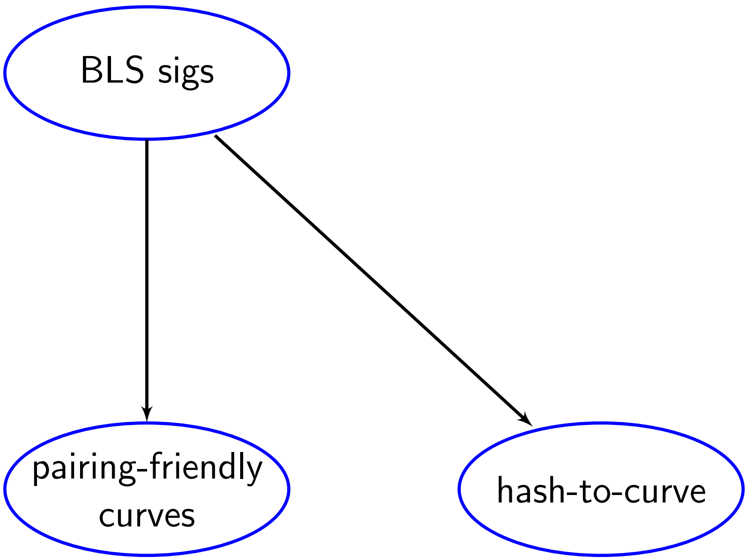
- Royalty-free "RAND-z" licenses are OK; commitments not to assert patents are better; unencumbered technologies are best.
- ➔ Don't patent crypto.

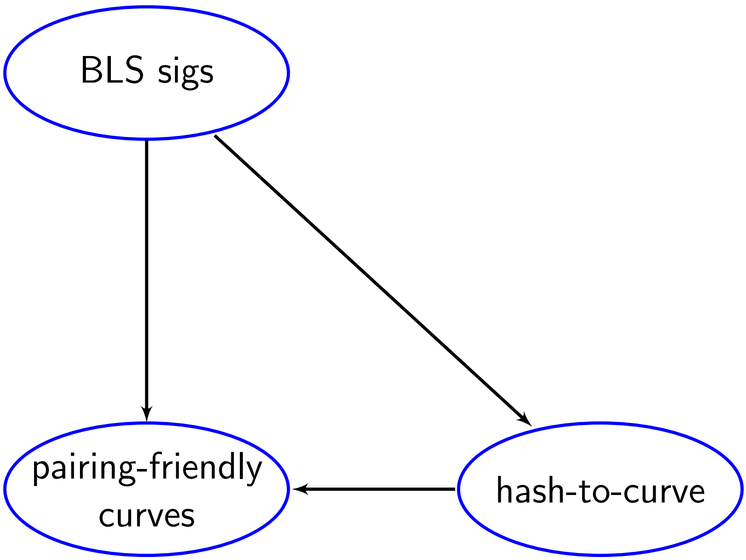
1. Standardizing advanced crypto with the IETF

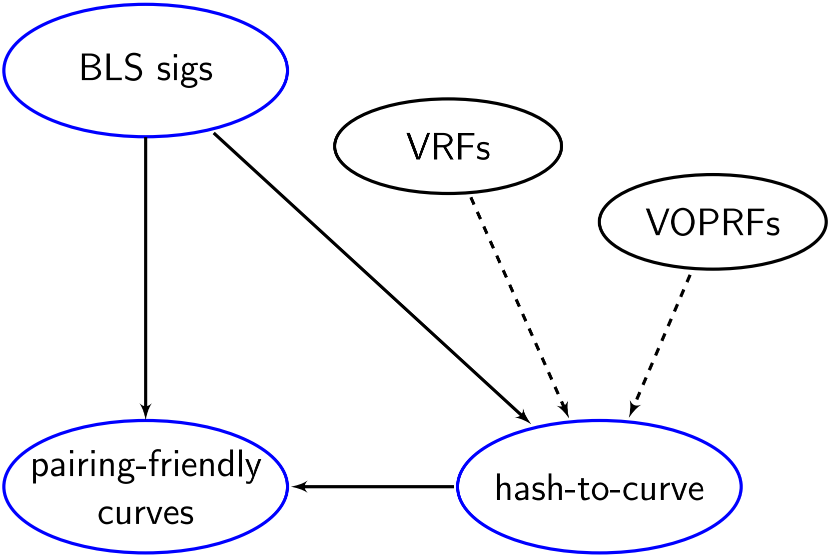
2. BLS signatures, hash-to-curve, and more


BLS sigs


BLS sigs
pairing-friendly
curves


BLS sigs
pairing-friendly
curves
hash-to-curve


BLS sigs
pairing-friendly curves
hash-to-curve


BLS sigs
VRFs
VOPRFs
pairing-friendly
curves
hash-to-curve

Pairing-friendly elliptic curves

A pairing-friendly elliptic curve defines:

- $\mathbb{G}_1 \subseteq E_1(\mathbb{F}_1)$ and $\mathbb{G}_2 \subseteq E_2(\mathbb{F}_2)$ of prime order q generated by P_1 and P_2, respectively
- $\mathbb{G}_T$ of prime order q

Pairing-friendly elliptic curves

A pairing-friendly elliptic curve defines:

- $\mathbb{G}_1 \subseteq E_1(\mathbb{F}_1)$ and $\mathbb{G}_2 \subseteq E_2(\mathbb{F}_2)$ of prime order q generated by P_1 and P_2, respectively
- $\mathbb{G}_T$ of prime order q
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, a bilinear map:

$$e(P_1^{\alpha}, P_2^{\beta}) = e(P_1, P_2)^{\alpha \cdot \beta} \qquad \alpha, \beta \in \mathbb{Z}_q$$

Pairing-friendly elliptic curves

A pairing-friendly elliptic curve defines:

- $\mathbb{G}_1 \subseteq E_1(\mathbb{F}_1)$ and $\mathbb{G}_2 \subseteq E_2(\mathbb{F}_2)$ of prime order q generated by P_1 and P_2, respectively
- $\mathbb{G}_T$ of prime order q
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, a bilinear map:

$$e(P_1^\alpha, P_2^\beta) = e(P_1, P_2)^{\alpha\cdot\beta} \qquad \alpha, \beta \in \mathbb{Z}_q$$

➔ What else might the spec cover?

Pairing-friendly elliptic curves

A pairing-friendly elliptic curve defines:

- $\mathbb{G}_1 \subseteq E_1(\mathbb{F}_1)$ and $\mathbb{G}_2 \subseteq E_2(\mathbb{F}_2)$ of prime order q generated by P_1 and P_2, respectively
- $\mathbb{G}_T$ of prime order q
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, a bilinear map:

$$e(P_1^{\alpha}, P_2^{\beta}) = e(P_1, P_2)^{\alpha \cdot \beta} \qquad \alpha, \beta \in \mathbb{Z}_q$$

➔ What else might the spec cover?

- serialization / deserialization

Pairing-friendly elliptic curves

A pairing-friendly elliptic curve defines:

- $\mathbb{G}_1 \subseteq E_1(\mathbb{F}_1)$ and $\mathbb{G}_2 \subseteq E_2(\mathbb{F}_2)$ of prime order q generated by P_1 and P_2, respectively
- $\mathbb{G}_T$ of prime order q
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, a bilinear map:

$$e(P_1^\alpha, P_2^\beta) = e(P_1, P_2)^{\alpha \cdot \beta} \qquad \alpha, \beta \in \mathbb{Z}_q$$

➔ What else might the spec cover?

- serialization / deserialization
- fast subgroup checks [Bowe19]

Pairing-friendly elliptic curves

A pairing-friendly elliptic curve defines:

- $\mathbb{G}_1 \subseteq E_1(\mathbb{F}_1)$ and $\mathbb{G}_2 \subseteq E_2(\mathbb{F}_2)$ of prime order q generated by P_1 and P_2, respectively
- $\mathbb{G}_T$ of prime order q
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, a bilinear map:

$$e(P_1^\alpha, P_2^\beta) = e(P_1, P_2)^{\alpha \cdot \beta} \qquad \alpha, \beta \in \mathbb{Z}_q$$

➔ What else might the spec cover?

- serialization / deserialization
- fast subgroup checks [Bowe19]
- test vectors

Hashing to elliptic curves (in constant time)

$\mathsf{HashToField}_i : \{0,1\}^\star \to \mathbb{F}$

a family of independent ROs indexed by i

Hashing to elliptic curves (in constant time)

$\mathsf{HashToField}_i : \{0,1\}^\star \to \mathbb{F}$

a family of independent ROs indexed by i

$\mathsf{MapToCurve} : \mathbb{F} \to E(\mathbb{F})$

[SvdW06,U07,Ica09,BCIMRT10,BHKL13,WB19]

Hashing to elliptic curves (in constant time)

$\mathsf{HashToField}_i : \{0,1\}^\star \to \mathbb{F}$

a family of independent ROs indexed by i

$\mathsf{MapToCurve} : \mathbb{F} \to E(\mathbb{F})$

[SvdW06,U07,Ica09,BCIMRT10,BHKL13,WB19]

$\mathsf{ClearCofactor} : E(\mathbb{F}) \to \mathbb{G}$

[SBCDK09,FKR11,BP18]

Hashing to elliptic curves (in constant time)

$\mathsf{HashToField}_i : \{0,1\}^\star \to \mathbb{F}$

a family of independent ROs indexed by i

$\mathsf{MapToCurve} : \mathbb{F} \to E(\mathbb{F})$

[SvdW06,U07,Ica09,BCIMRT10,BHKL13,WB19]

$\mathsf{ClearCofactor} : E(\mathbb{F}) \to \mathbb{G}$

[SBCDK09,FKR11,BP18]

$H(\mathsf{msg}) \to \mathbb{G}$ [BCIMRT10,FFSTV13]

$$Q_1 = \mathsf{MapToCurve}(\mathsf{HashToField}_1(\mathsf{msg}))$$
$$Q_2 = \mathsf{MapToCurve}(\mathsf{HashToField}_2(\mathsf{msg}))$$
$$\text{output } \mathsf{ClearCofactor}(Q_1 \cdot Q_2)$$

is indifferentiable from a random oracle to $\mathbb{G}$

Hashing to elliptic curves (in constant time)

$\mathsf{HashToField}_i : \{0,1\}^\star \to \mathbb{F}$
a family of independent ROs indexed by i

$\mathsf{MapToCurve} : \mathbb{F} \to E(\mathbb{F})$
[SvdW06,U07,Ica09,BCIMRT10,BHKL13,WB19]

$\mathsf{ClearCofactor} : E(\mathbb{F}) \to \mathbb{G}$
[SBCDK09,FKR11,BP18]

$H(\mathsf{msg}) \to \mathbb{G}$ [BCIMRT10,FFSTV13]

$$Q_1 = \mathsf{MapToCurve}(\mathsf{HashToField}_1(\mathsf{msg}))$$
$$Q_2 = \mathsf{MapToCurve}(\mathsf{HashToField}_2(\mathsf{msg}))$$
$$\text{output } \mathsf{ClearCofactor}(Q_1 \cdot Q_2)$$

is indifferentiable from a random oracle to $\mathbb{G}$

BLS signatures [BLS01]

KeyGen() $\to (pk, sk)$: $x \xleftarrow{R} \mathbb{Z}_q$; output (P_2^x, x).

BLS signatures [BLS01]

KeyGen() $\rightarrow (pk, sk)$: $x \xleftarrow{R} \mathbb{Z}_q$; output (P_2^x, x).

Sign(sk, msg) $\rightarrow$ sig: output $H(\text{msg})^{sk} \in \mathbb{G}_1$.

BLS signatures [BLS01]

KeyGen() $\rightarrow (pk, sk)$: $x \xleftarrow{\text{R}} \mathbb{Z}_q$; output (P_2^x, x).

Sign$(sk, \text{msg}) \rightarrow \text{sig}$: output $H(\text{msg})^{sk} \in \mathbb{G}_1$.

Verify$(pk, \text{msg}, \text{sig}) \rightarrow \{\text{OK}, \perp\}$:
if $e(H(\text{msg}), pk) = e(\text{sig}, P_2)$, output OK,
else output $\perp$.

BLS signatures [BLS01]

KeyGen() $\to (pk, sk)$: $x \xleftarrow{\mathrm{R}} \mathbb{Z}_q$; output (P_2^x, x).

Sign(sk, msg) $\to$ sig: output $H(\mathrm{msg})^{sk} \in \mathbb{G}_1$.

Verify(pk, msg, sig) $\to \{\mathrm{OK}, \perp\}$:
if $e(H(\mathrm{msg}), pk) = e(\mathrm{sig}, P_2)$, output OK,
else output $\perp$.

$$e(H(\mathrm{msg}), P_2^x) = e(H(\mathrm{msg})^x, P_2)$$

BLS signatures [BLS01]

KeyGen() $\to (pk, sk)$: $x \xleftarrow{R} \mathbb{Z}_q$; output (P_2^x, x).

Sign$(sk, \text{msg}) \to \text{sig}$: output $H(\text{msg})^{sk} \in \mathbb{G}_1$.

Verify$(pk, \text{msg}, \text{sig}) \to \{\text{OK}, \perp\}$:
if $e(H(\text{msg}), pk) = e(\text{sig}, P_2)$, output OK,
else output $\perp$.

$$e(H(\text{msg}), P_2^x) = e(H(\text{msg})^x, P_2)$$

BLS signatures [BLS01]

KeyGen() $\to$ (pk, sk) $x \xleftarrow{R} \mathbb{Z}_q$; output (P_2^x, x).

Sign(sk, msg) $\to$ sig: output $H(\text{msg})^{sk} \in \mathbb{G}_1$.

Verify$(pk, \text{msg}, \text{sig})$ $\to \{\text{OK}, \perp\}$:
if $e(H(\text{msg}), pk) = e(\text{sig}, P_2)$, output OK,
else output $\perp$.

$$e(H(\text{msg}), P_2^x) = e(H(\text{msg})^x, P_2)$$

BLS signature aggregation [BGLS03]

$\text{Aggregate}(\text{sig}_1, \ldots, \text{sig}_n) \to \text{sig}$:

output $\prod_i \text{sig}_i \in \mathbb{G}_1$.

BLS signature aggregation [BGLS03]

Aggregate($\mathsf{sig}_1, \ldots, \mathsf{sig}_n$) $\rightarrow$ sig:

output $\prod_i \mathsf{sig}_i \in \mathbb{G}_1$.

VerMulti($pk_1, \ldots, pk_n$, msg, sig) $\rightarrow \{\mathrm{OK}, \perp\}$:

if $e(H(msg), \prod_i pk_i) = e(\mathsf{sig}, P_2)$, output OK,
else output $\perp$.

BLS signature aggregation [BGLS03]

Aggregate($\mathsf{sig}_1, \ldots, \mathsf{sig}_n$) $\to$ sig:
output $\prod_i \mathsf{sig}_i \in \mathbb{G}_1$.

VerMulti($pk_1, \ldots, pk_n, \mathsf{msg}, sig$) $\to \{\mathsf{OK}, \bot\}$:
if $e(H(msg), \prod_i pk_i) = e(\mathsf{sig}, P_2)$, output OK,
else output $\bot$.

VerBatch($pk_1, \mathsf{msg}_1, \ldots, pk_n, \mathsf{msg}_n, \mathsf{sig}$) $\to \{\mathsf{OK}, \bot\}$:
if $\prod_i e(H(\mathsf{msg}_i), pk_i) = e(\mathsf{sig}, P_2)$, output OK,
else output $\bot$.

BLS signature aggregation [BGLS03]

$\text{Aggregate}(\text{sig}_1, \ldots, \text{sig}_n) \to \text{sig}$:
output $\prod_i \text{sig}_i \in \mathbb{G}_1$.

$\text{VerMulti}(pk_1, \ldots, pk_n, \text{msg}, sig) \to \{\text{OK}, \perp\}$:
if $e(H(msg), \prod_i pk_i) = e(\text{sig}, P_2)$, output OK,
else output $\perp$.

$\text{VerBatch}(pk_1, \text{msg}_1, \ldots, pk_n, \text{msg}_n, \text{sig}) \to \{\text{OK}, \perp\}$:
if $\prod_i e(H(\text{msg}_i), pk_i) = e(\text{sig}, P_2)$, output OK,
else output $\perp$.

Rogue key attack

Let's say Alice has pk_a and Bob has pk_b.

Mallory samples $x \xleftarrow{R} \mathbb{Z}_q$ and computes

$$pk_m = P_2^x \cdot (pk_a \cdot pk_b)^{-1}$$

Rogue key attack

Let's say Alice has pk_a and Bob has pk_b.

Mallory samples $x \xleftarrow{\mathrm{R}} \mathbb{Z}_q$ and computes

$$pk_m = P_2^x \cdot (pk_a \cdot pk_b)^{-1}$$

Since $\prod pk_i = P_2^x$, Mallory can forge a multi-signature for any msg:

$$e(H(\text{msg}), \prod pk_i) = e(H(\text{msg})^x, P_2)$$

Defending against rogue keys

Require unique messages [BGLS03]:

But: no fast multi-sig verification.

Defending against rogue keys

Require unique messages [BGLS03]:

But: no fast multi-sig verification.

Message augmentation [BGLS03,BNN07]:

Sign $pk \,||\, \text{msg}$, ensuring uniqueness
(so: no fast multi-sig verification).

Defending against rogue keys

Require unique messages [BGLS03]:

But: no fast multi-sig verification.

Message augmentation [BGLS03,BNN07]:

Sign $pk \,||\, \mathrm{msg}$, ensuring uniqueness
(so: no fast multi-sig verification).

Proof of possession [Bol03,LOSSW06,RY07]:

Require key owners to furnish $\mathrm{Sign}(sk, pk)$
(gives fast multi-sig verification).

Defending against rogue keys

Require unique messages [BGLS03]:

But: no fast multi-sig verification.

Message augmentation [BGLS03,BNN07]:

Sign $pk \,||\, \text{msg}$, ensuring uniqueness
(so: no fast multi-sig verification).

Proof of possession [Bol03,LOSSW06,RY07]:

Require key owners to furnish $\text{Sign}(sk, pk)$
(gives fast multi-sig verification).

Random linear combination [BDN18]:

Check $e(H(\text{msg}_i), \prod_i pk_i^{\alpha_i}) \stackrel{?}{=} e(\prod_i \text{sig}_i^{\alpha_i}, P_2)$
for pseudorandomly-generated α_i.

Defending against rogue keys

Require unique messages [BGLS03]:

But: no fast multi-sig verification.

Message augmentation [BGLS03,BNN07]:

Sign $pk \,||\, \text{msg}$, ensuring uniqueness
(so: no fast multi-sig verification).

Proof of possession [Bol03,LOSSW06,RY07]:

Require key owners to furnish $\text{Sign}(sk, pk)$
(gives fast multi-sig verification).

Random linear combination [BDN18]:

Check $e(H(\text{msg}_i), \prod_i pk_i^{\alpha_i}) \stackrel{?}{=} e(\prod_i \text{sig}_i^{\alpha_i}, P_2)$
for pseudorandomly-generated α_i.

Lessons learned (so far)

✓ Make things hard to break, but add firewalls for when they do.

Lessons learned (so far)

- ✓ Make things hard to break, but add firewalls for when they do.
- ✓ Implement, implement, implement: that's what the standard is for!

Lessons learned (so far)

- ✓ Make things hard to break, but add firewalls for when they do.
- ✓ Implement, implement, implement: that's what the standard is for!
- ✓ You can't make everyone happy (but don't take it personally).

Lessons learned (so far)

- ✓ Make things hard to break, but add firewalls for when they do.
- ✓ Implement, implement, implement: that's what the standard is for!
- ✓ You can't make everyone happy (but don't take it personally).
- ✓ The IETF is a great place for new crypto!

Lessons learned (so far)

- ✓ Make things hard to break, but add firewalls for when they do.
- ✓ Implement, implement, implement: that's what the standard is for!
- ✓ You can't make everyone happy (but don't take it personally).
- ✓ The IETF is a great place for new crypto!

`https://github.com/cfrg/draft-irtf-cfrg-bls-signature`
`https://github.com/cfrg/draft-irtf-cfrg-hash-to-curve`
`https://bls-hash.crypto.fyi`