Universal Signature Aggregators

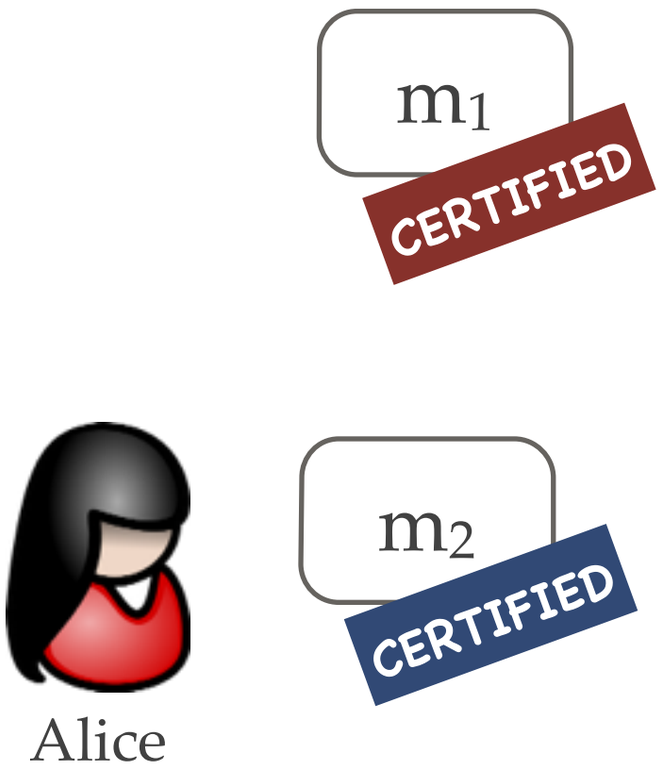
Susan Hohenberger (JHU)
Venkata Koppula (UT Austin)
Brent Waters (UT Austin)

Signature Aggregation (BGLS-03)

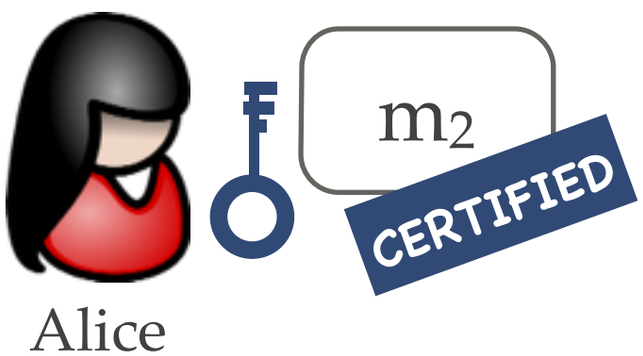
Signature Aggregation (BGLS-03)

Alice

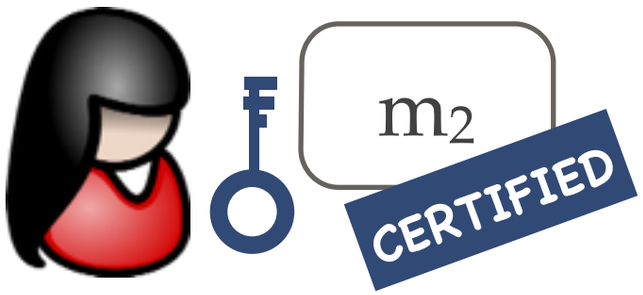
Signature Aggregation (BGLS-03)



Signature Aggregation (BGLS-03)

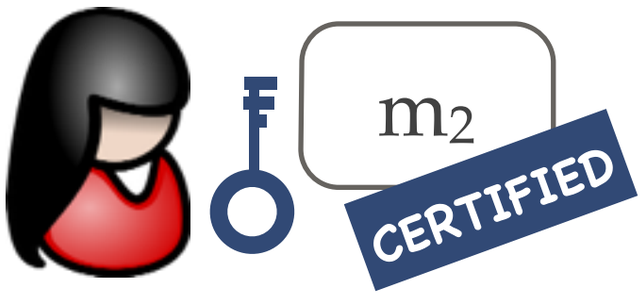


Signature Aggregation (BGLS-03)



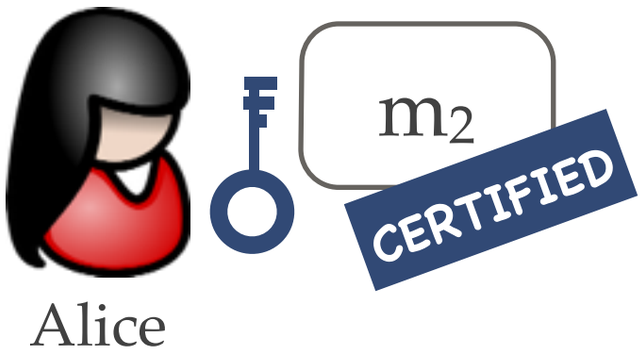
Alice

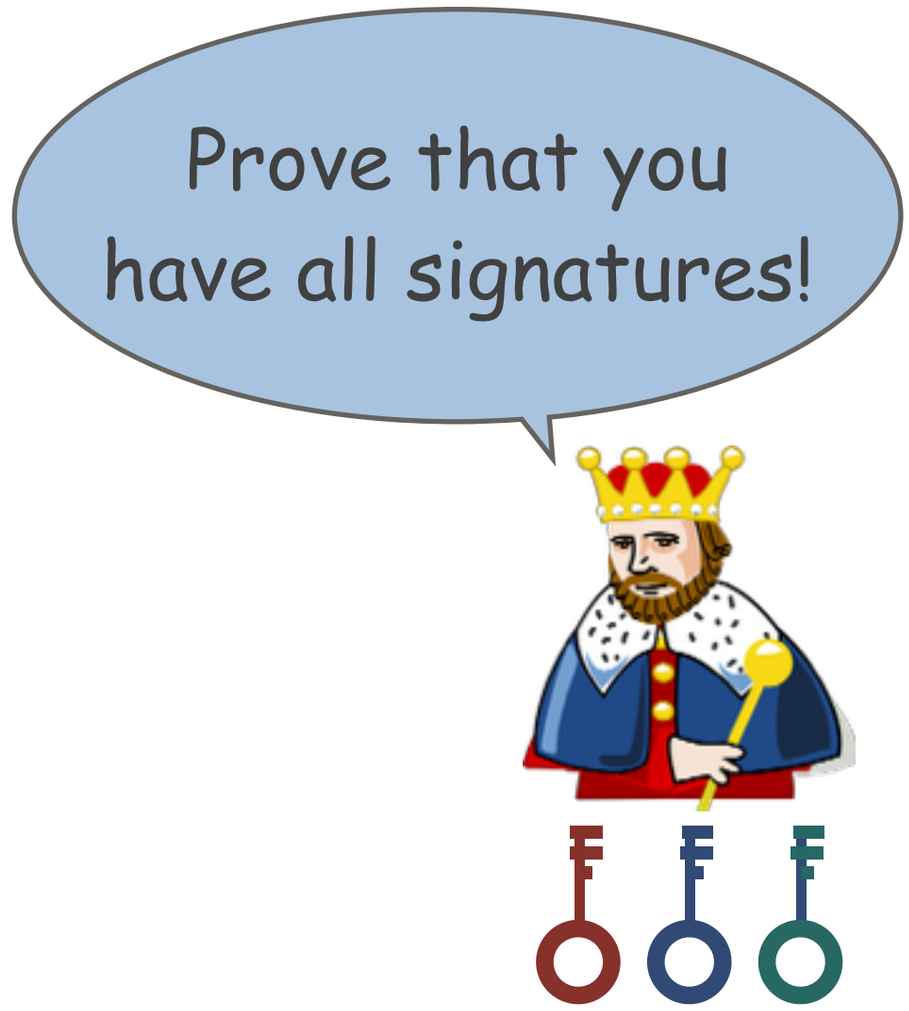
Signature Aggregation (BGLS-03)



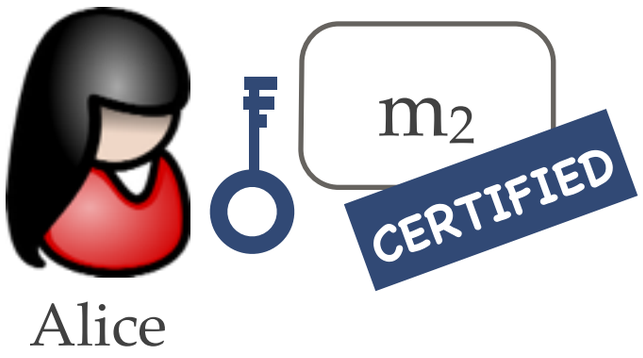
Alice

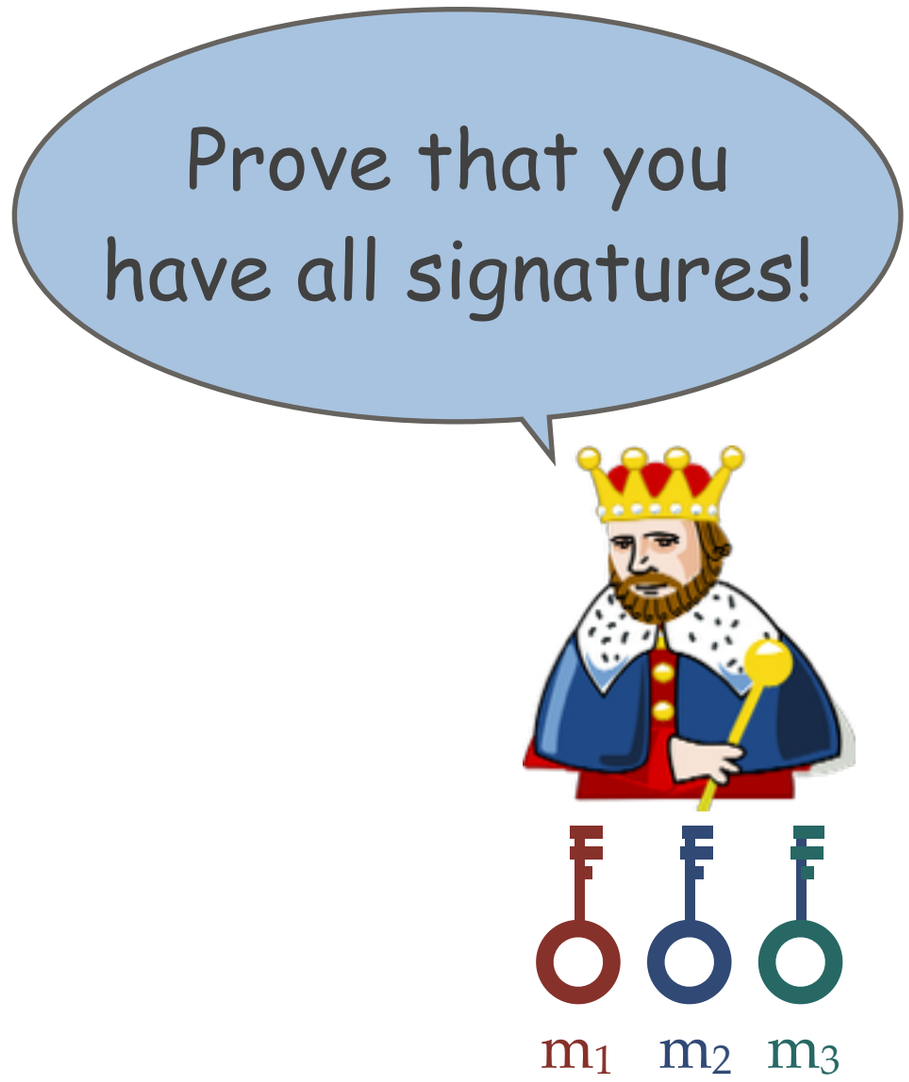
Signature Aggregation (BGLS-03)



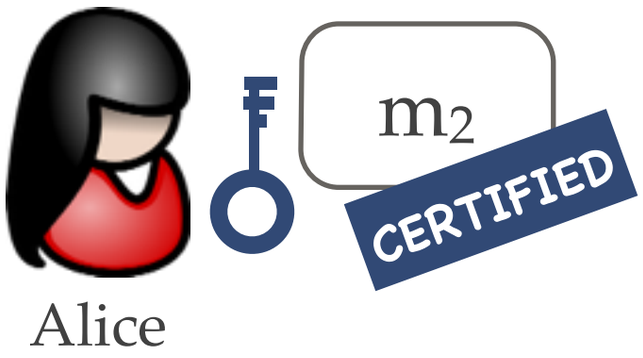


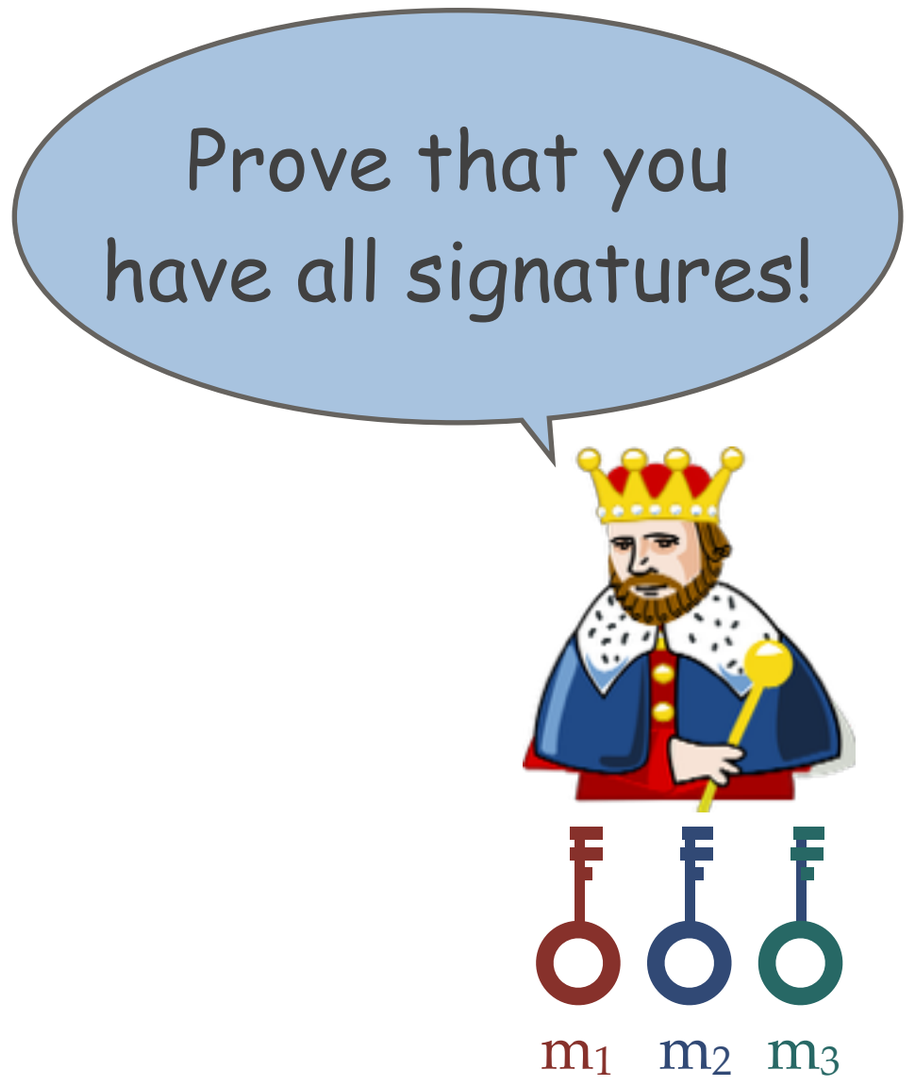
Signature Aggregation (BGLS-03)



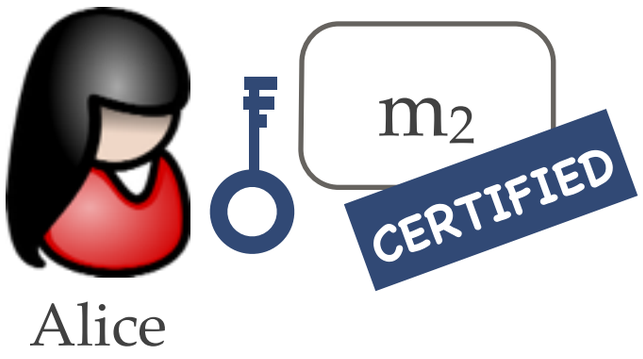


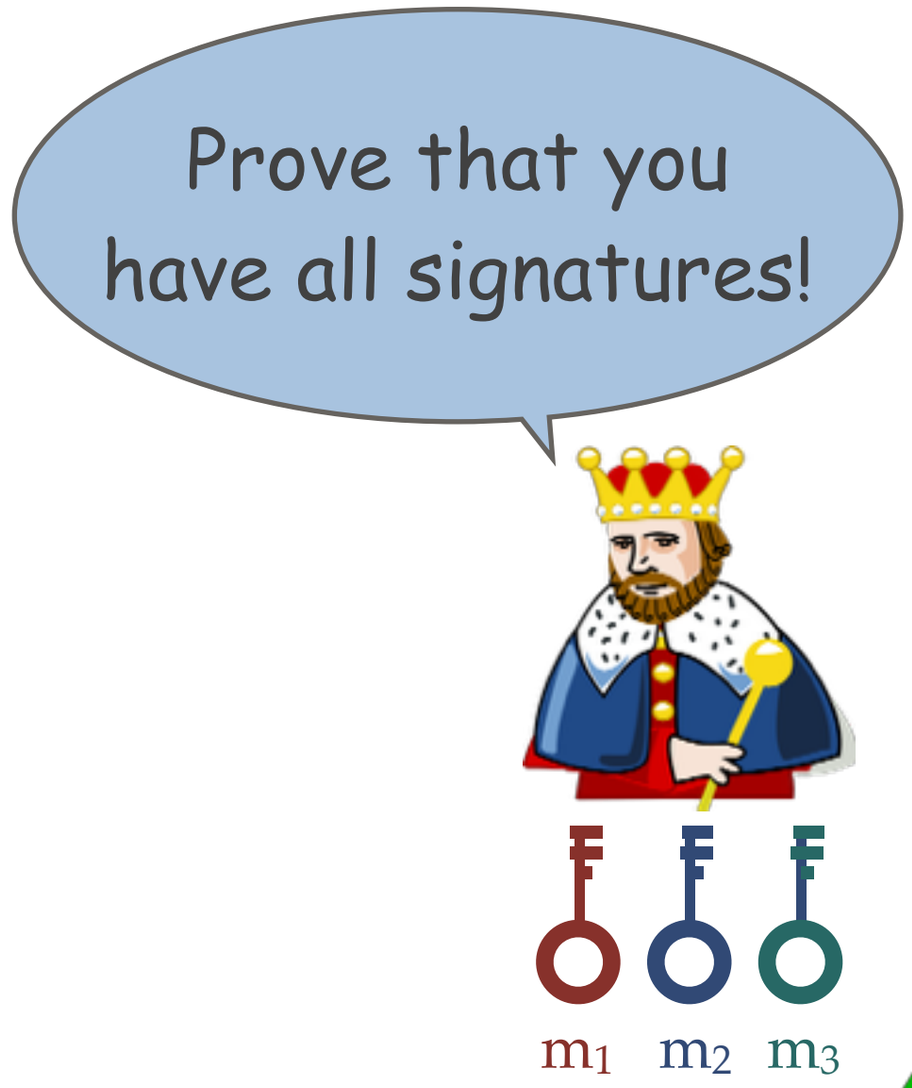
Signature Aggregation (BGLS-03)

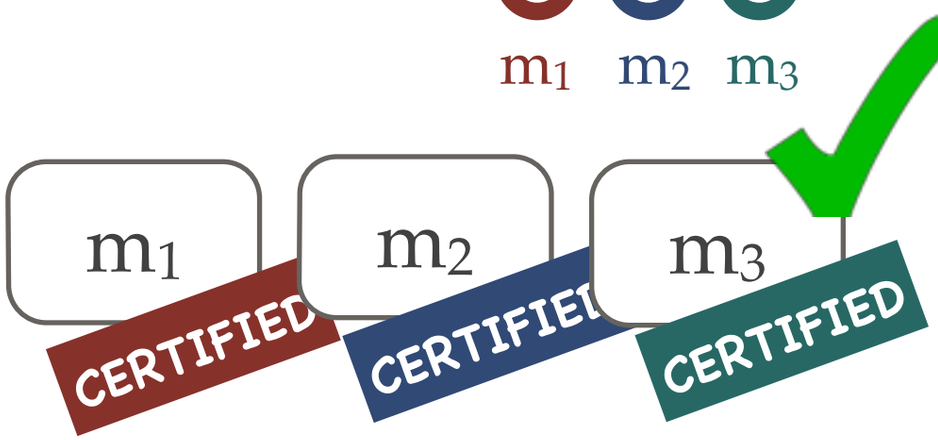




Signature Aggregation (BGLS-03)

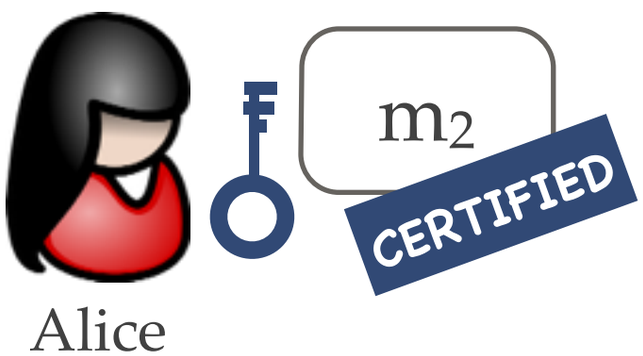






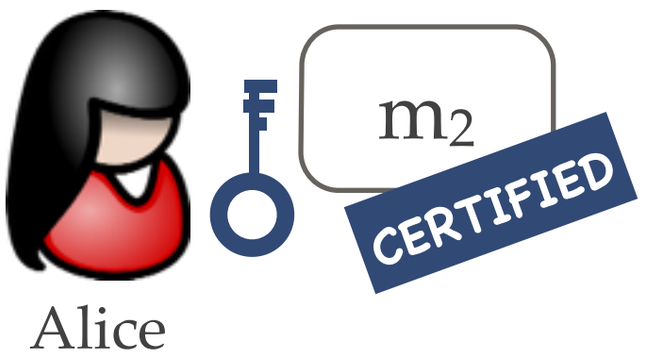
Signature Aggregation (BGLS-03)

Aggregate Signatures
Boneh, Gentry, Lynn, Shacham
Eurocrypt 03



Signature Aggregation (BGLS-03)

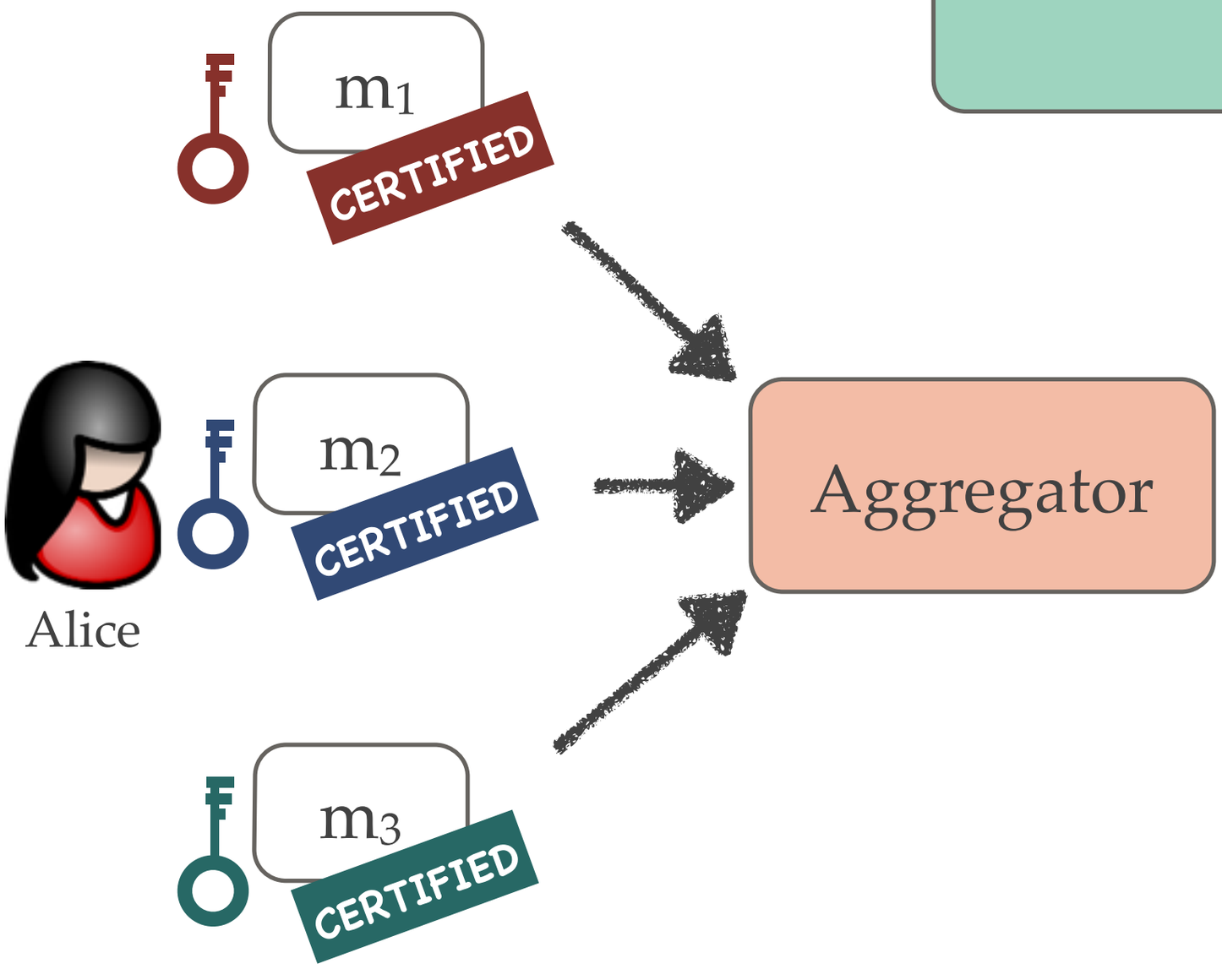
Aggregate Signatures
Boneh, Gentry, Lynn, Shacham
Eurocrypt 03



Aggregator

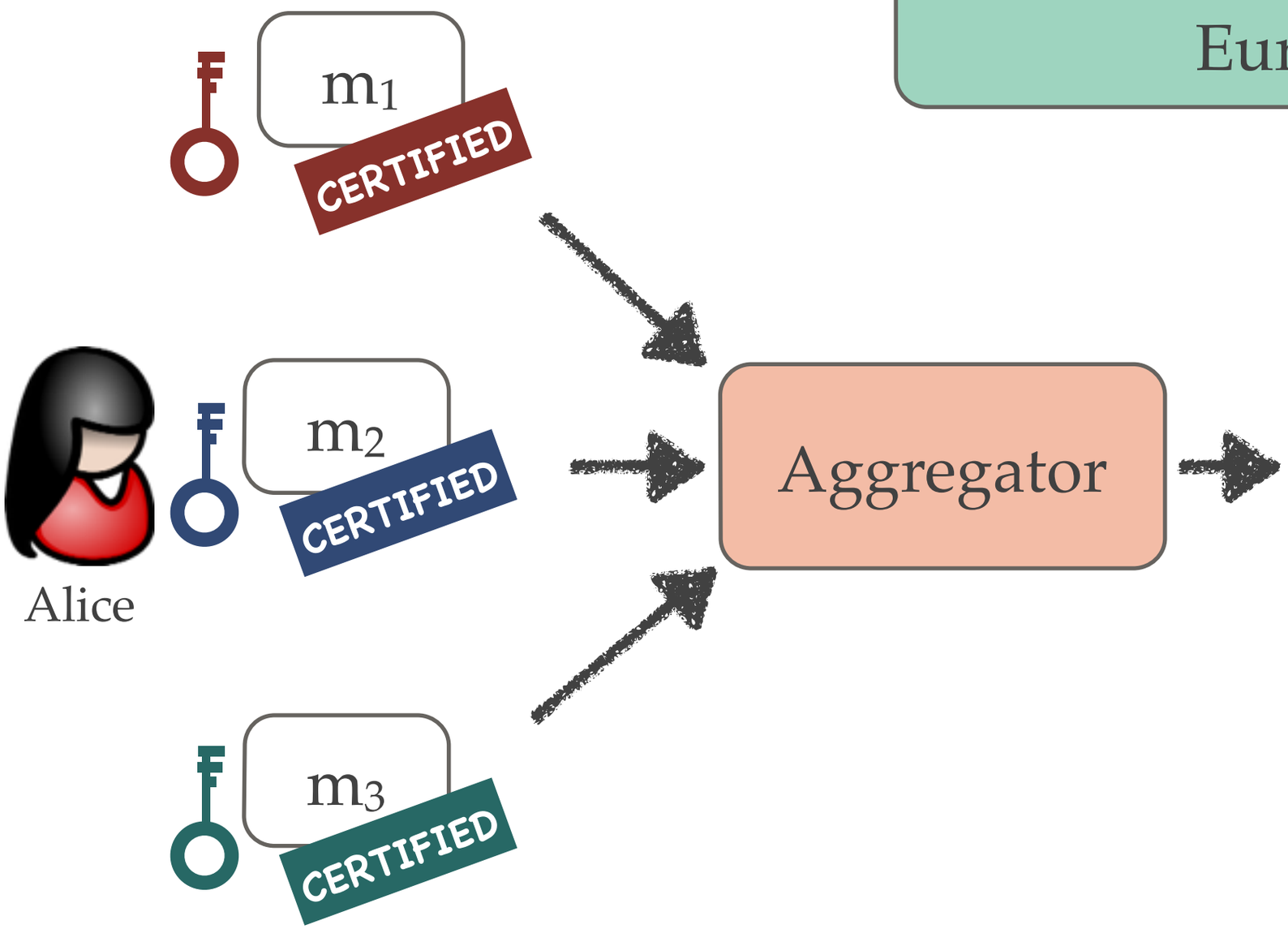
Signature Aggregation (BGLS-03)

Aggregate Signatures
Boneh, Gentry, Lynn, Shacham
Eurocrypt 03



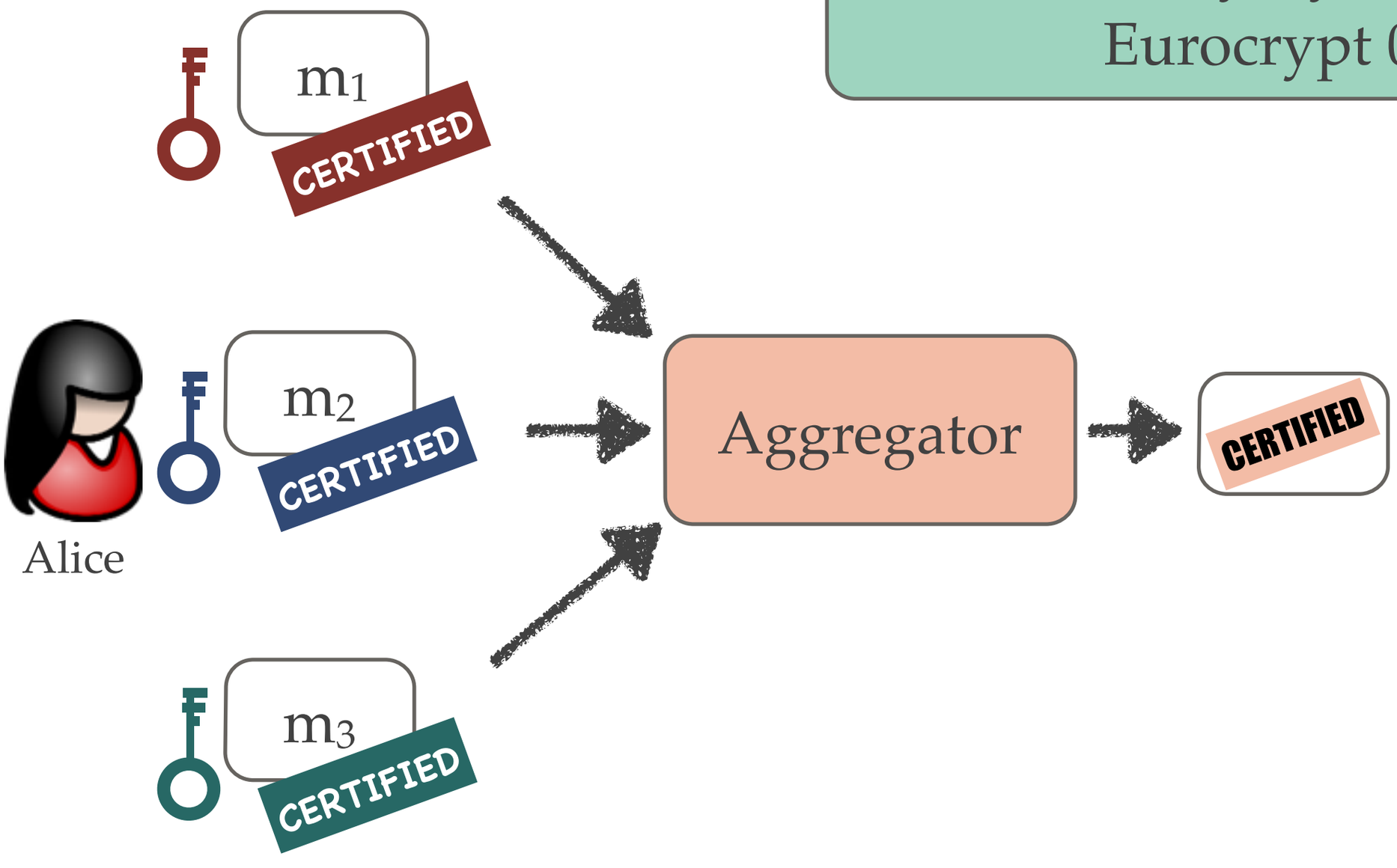
Signature Aggregation (BGLS-03)

Aggregate Signatures
Boneh, Gentry, Lynn, Shacham
Eurocrypt 03



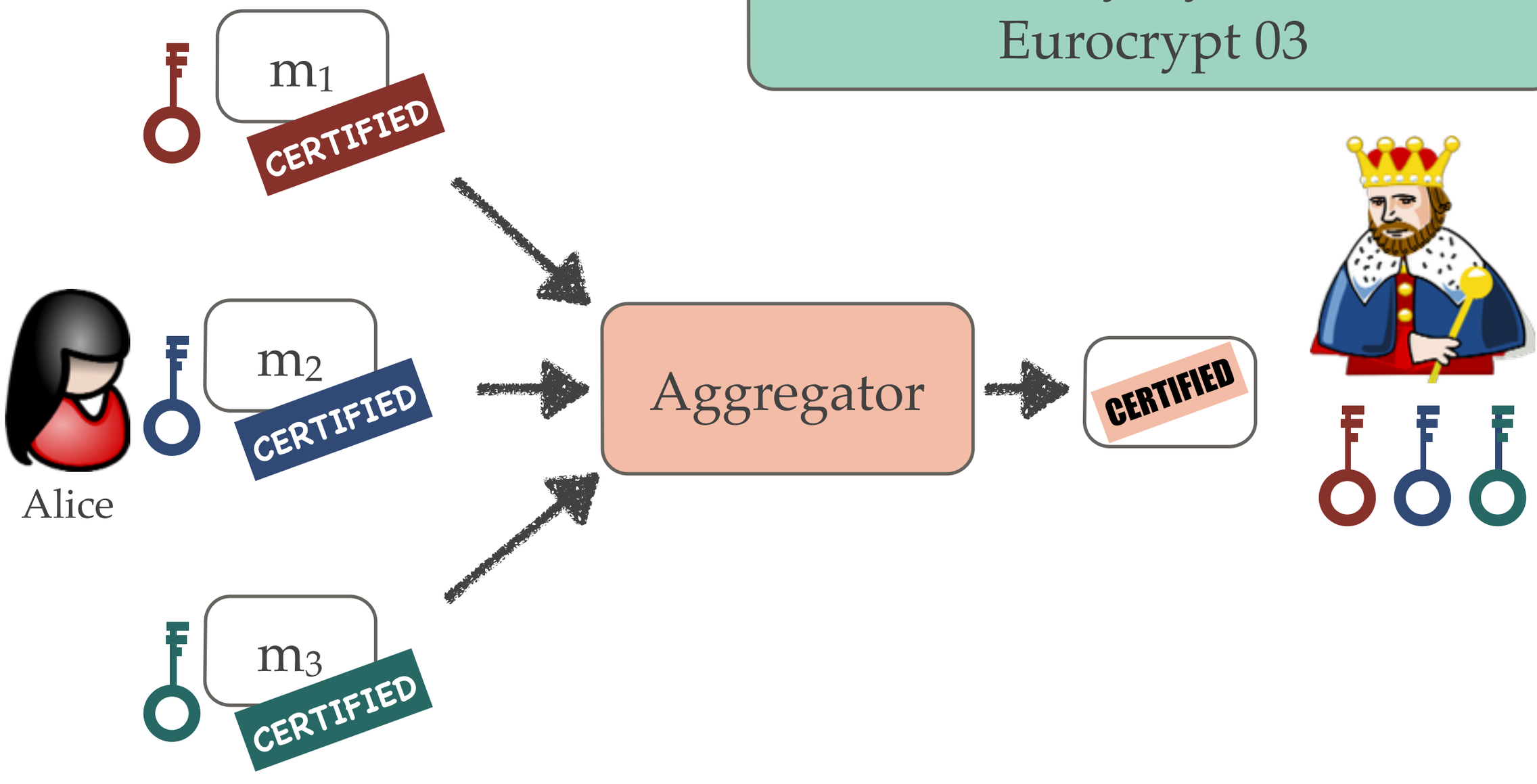
Signature Aggregation (BGLS-03)

Aggregate Signatures
Boneh, Gentry, Lynn, Shacham
Eurocrypt 03



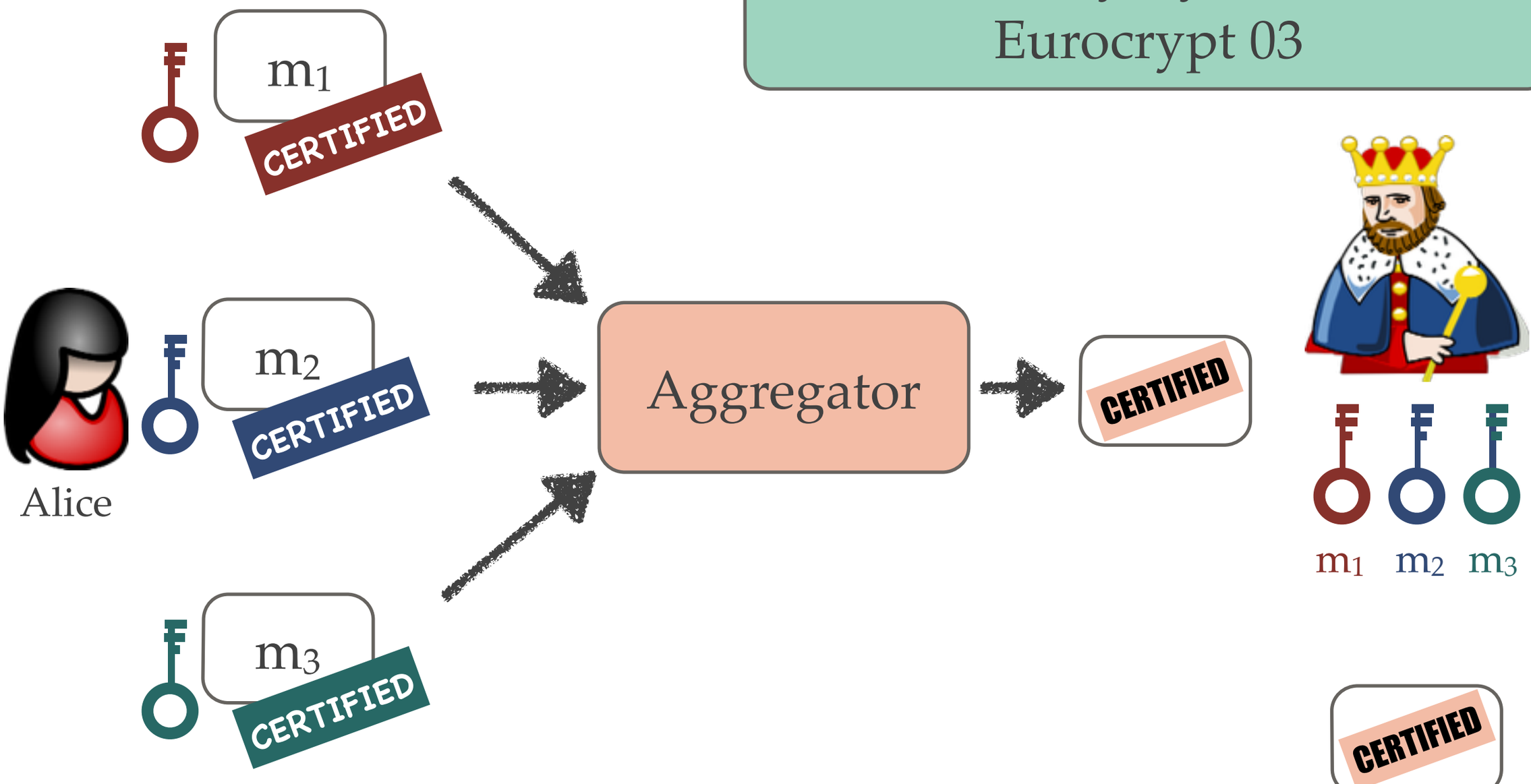
Signature Aggregation (BGLS-03)

Aggregate Signatures
Boneh, Gentry, Lynn, Shacham
Eurocrypt 03



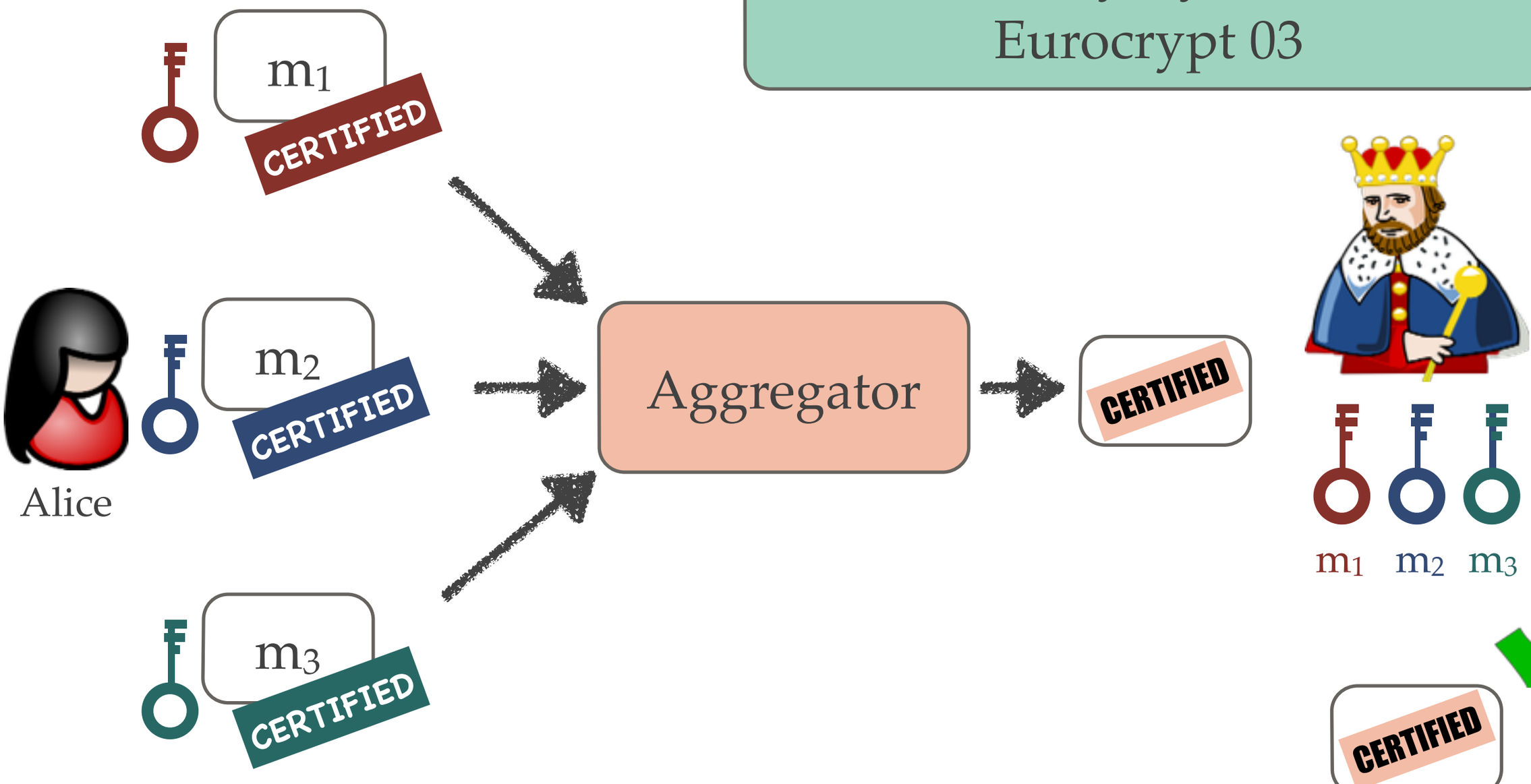
Signature Aggregation (BGLS-03)

Aggregate Signatures
Boneh, Gentry, Lynn, Shacham
Eurocrypt 03

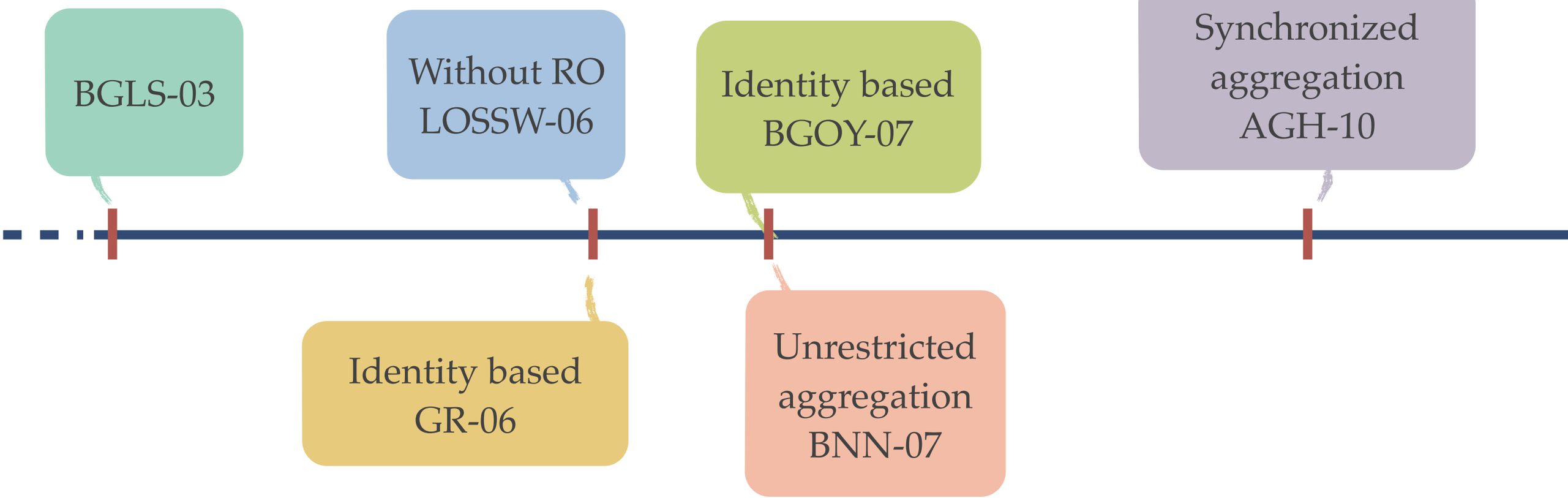


Signature Aggregation (BGLS-03)

Aggregate Signatures
Boneh, Gentry, Lynn, Shacham
Eurocrypt 03

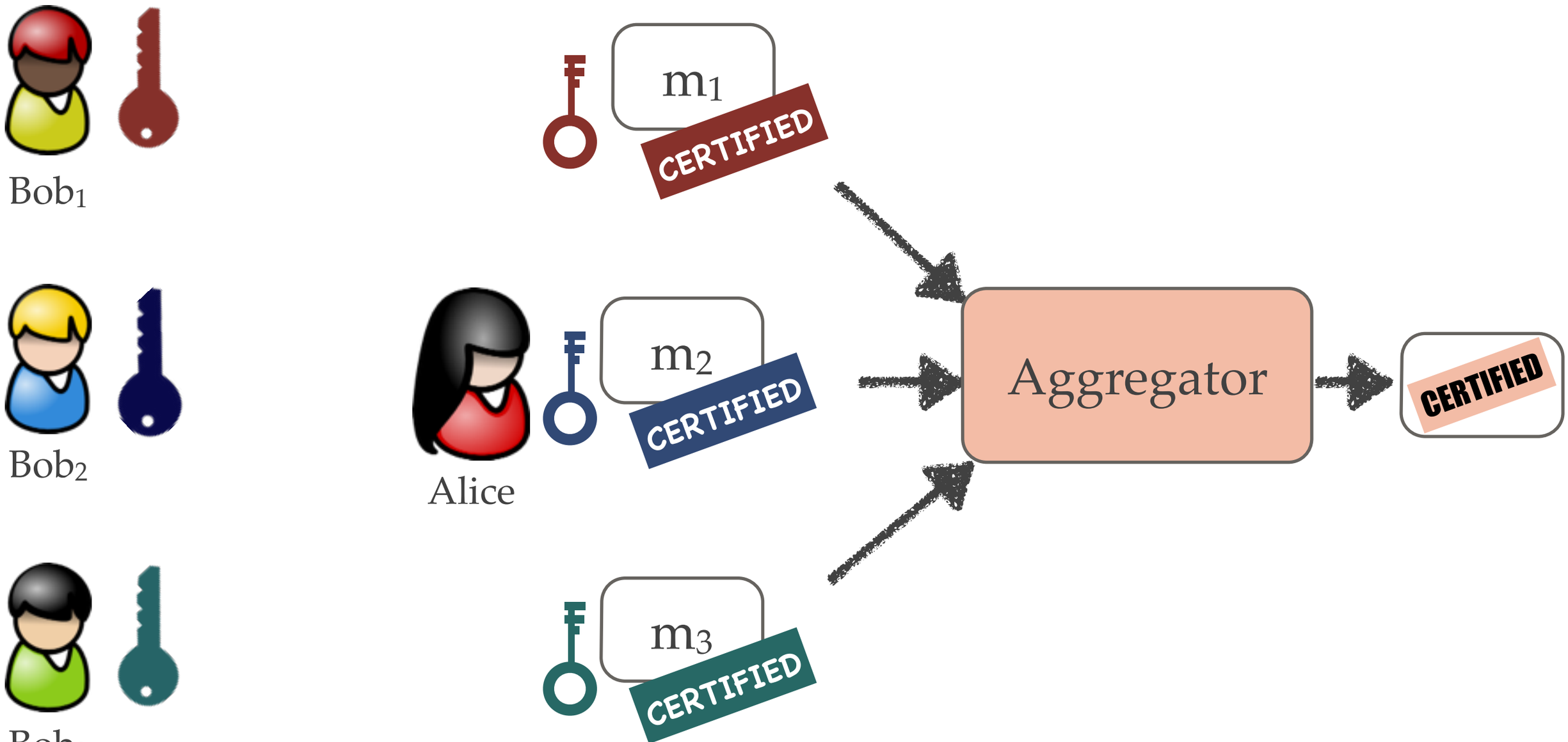


Signature Aggregation (BGLS-03)

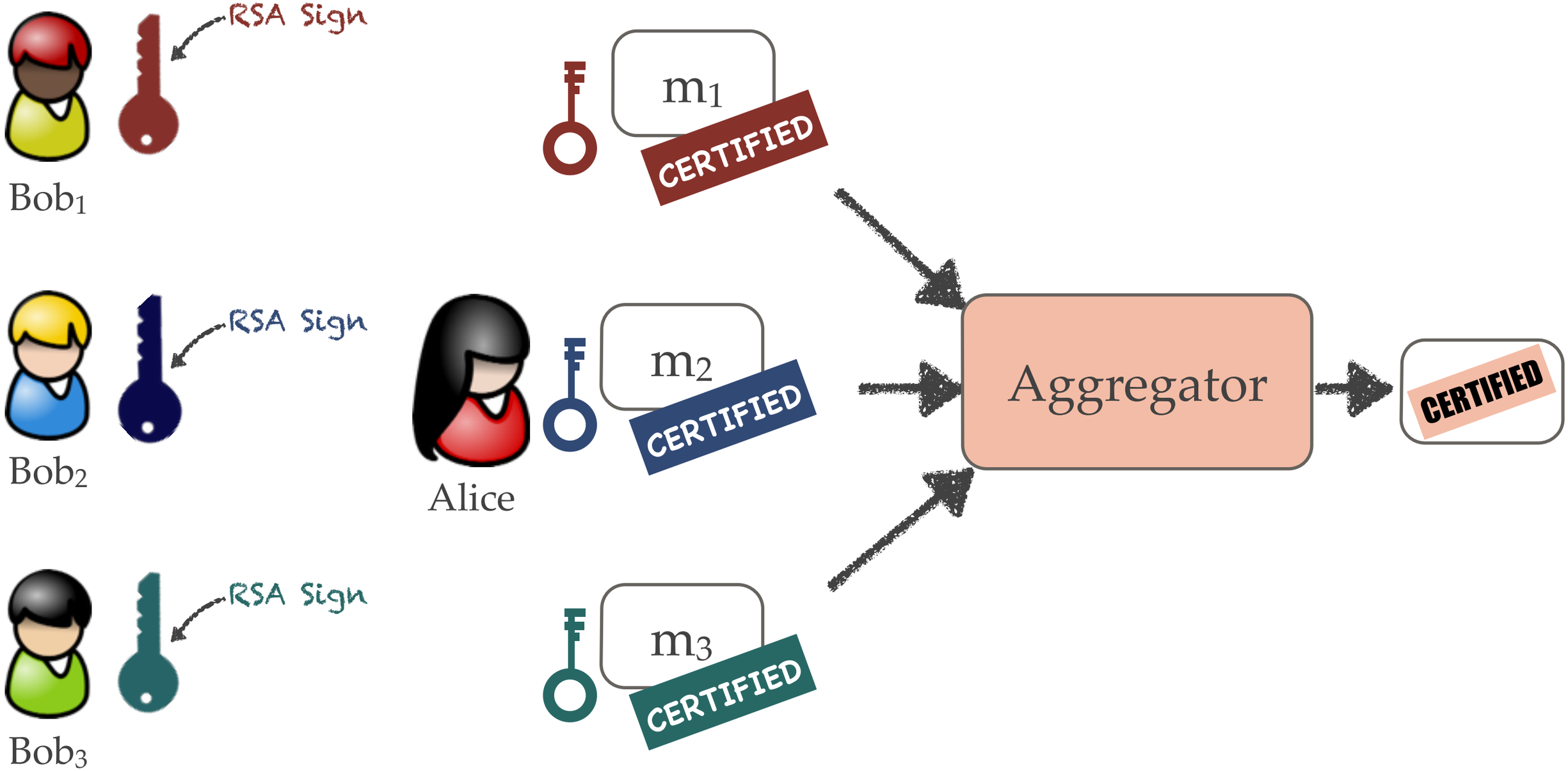


All signers must adopt a common signature system that allows aggregation.

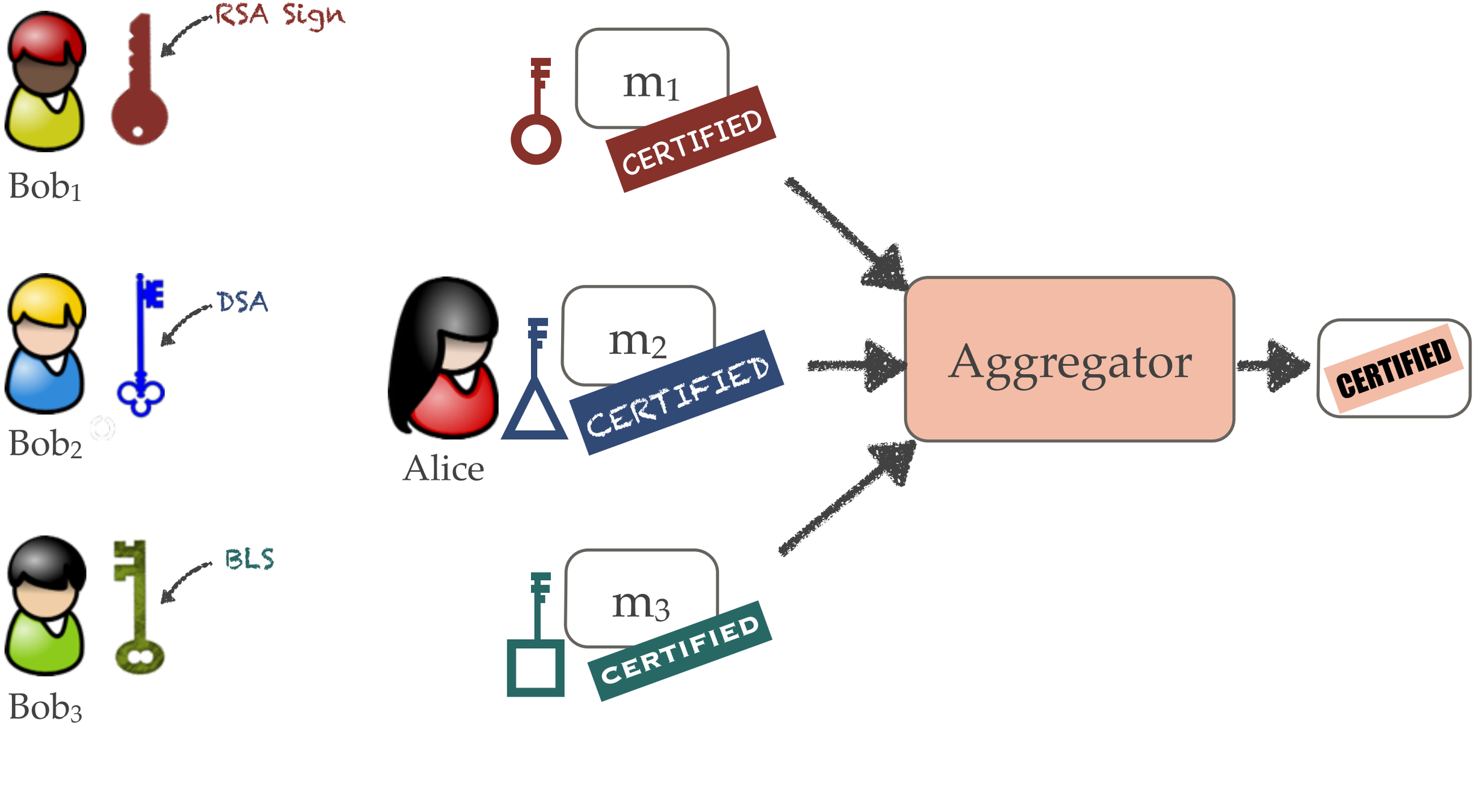
Limitations of Existing Schemes



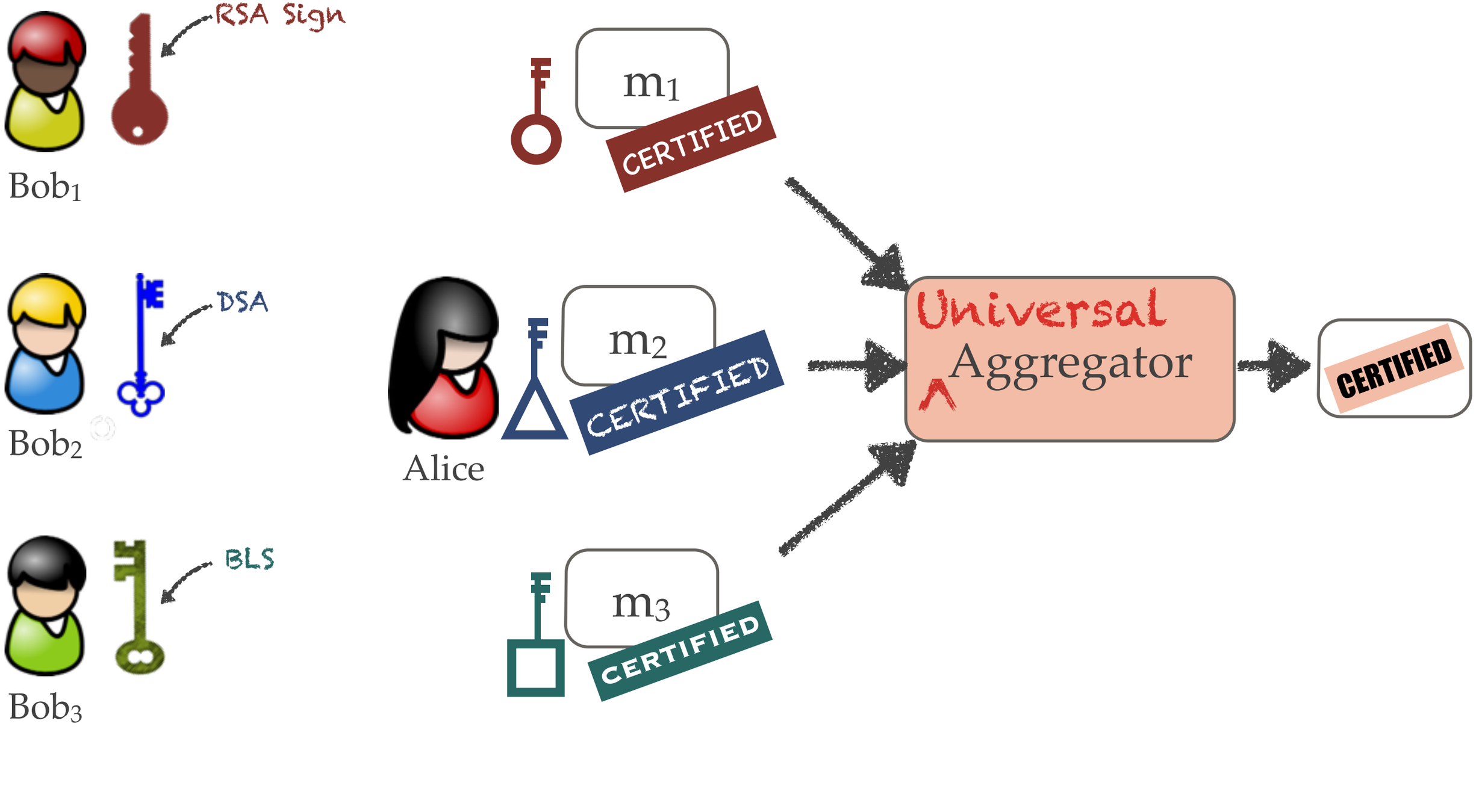
Limitations of Existing Schemes



Limitations of Existing Schemes



Limitations of Existing Schemes



?

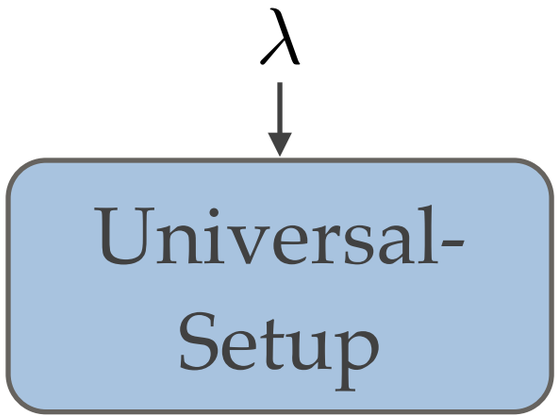
Can we have a `Universal Aggregator' that can aggregate signatures from a `large' class of signature schemes?

Universal Signature Aggregators

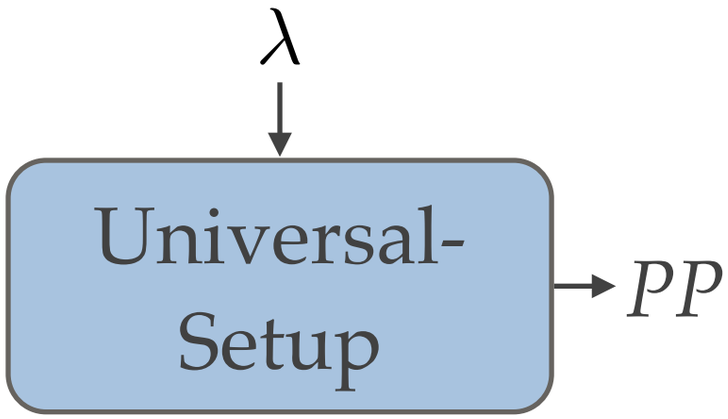
Universal Signature Aggregators

Universal-Setup

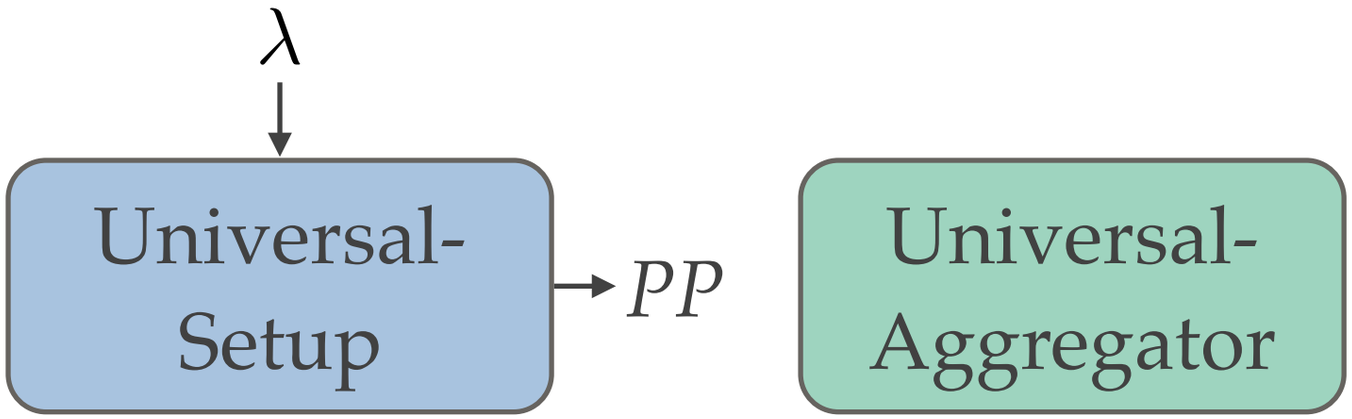
Universal Signature Aggregators



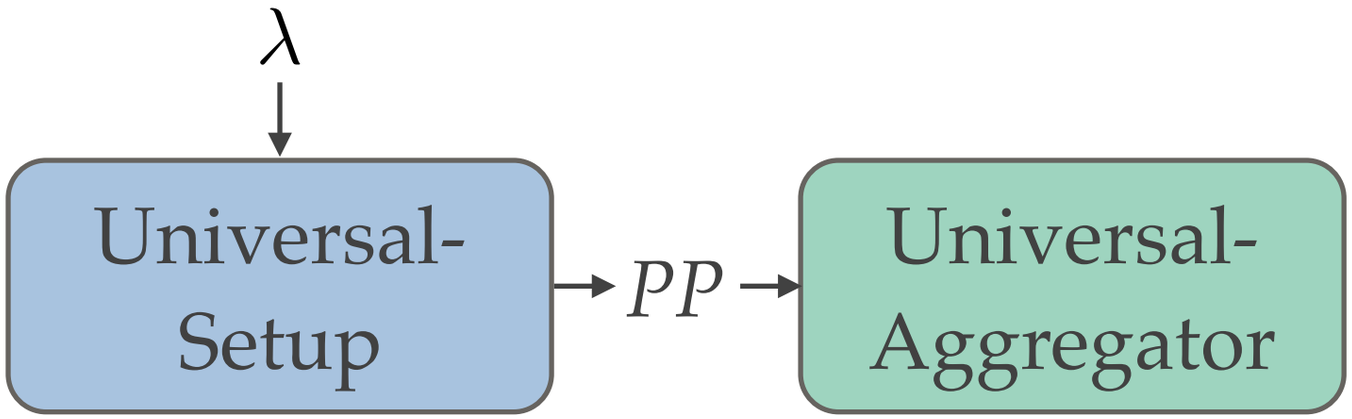
Universal Signature Aggregators



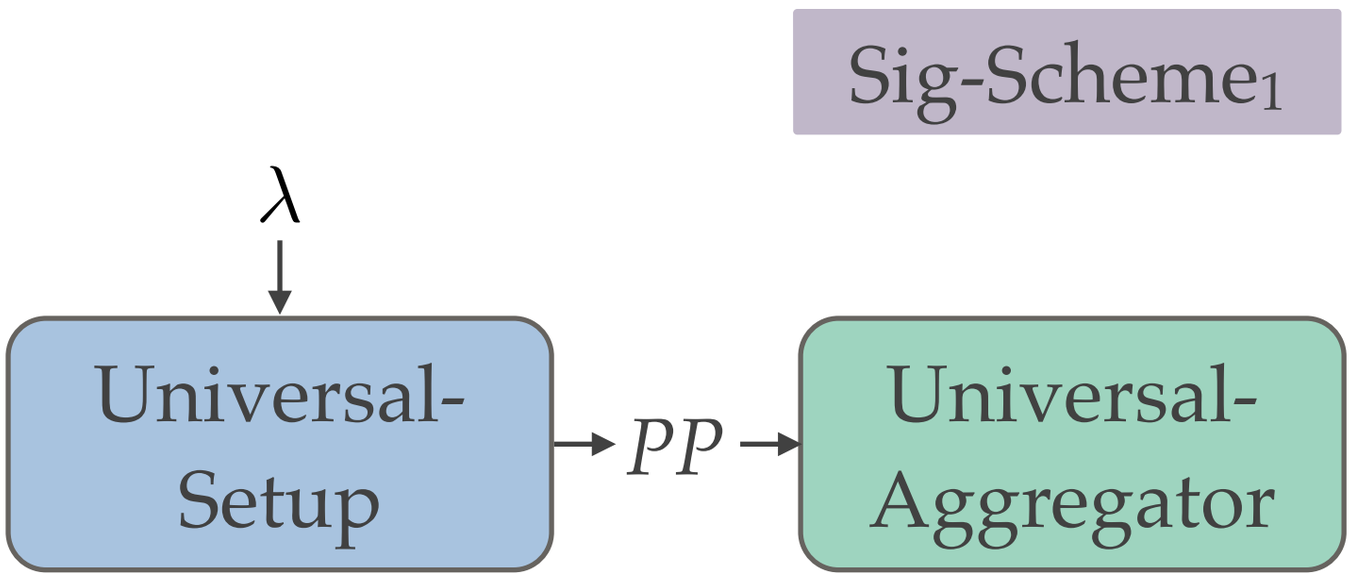
Universal Signature Aggregators



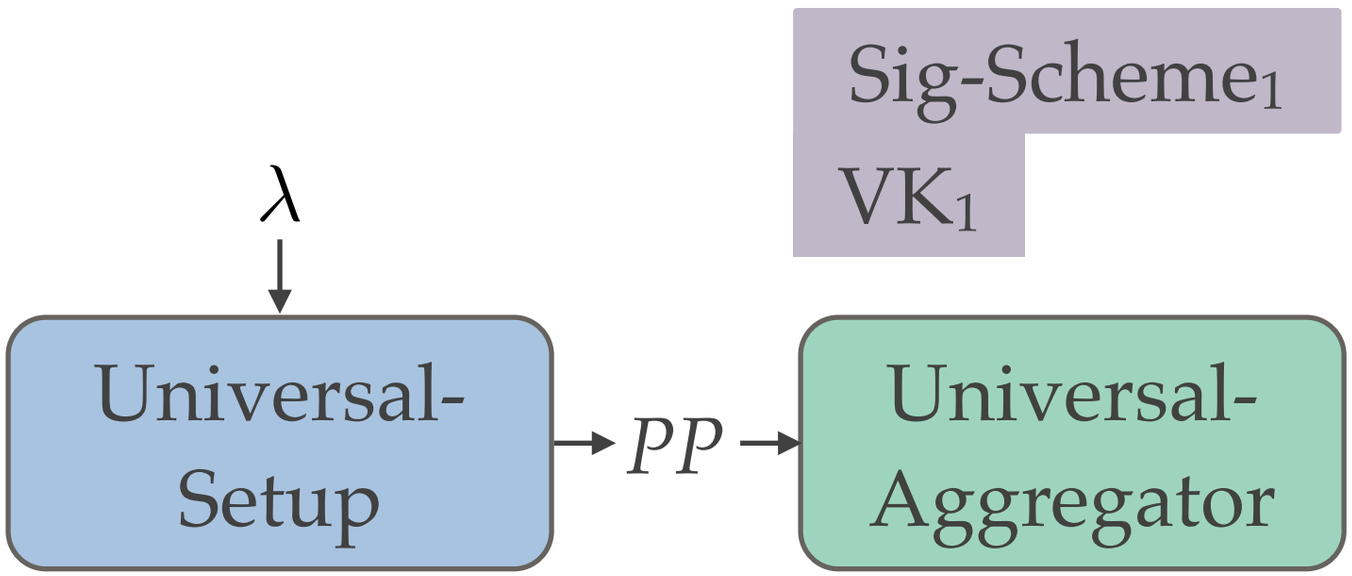
Universal Signature Aggregators



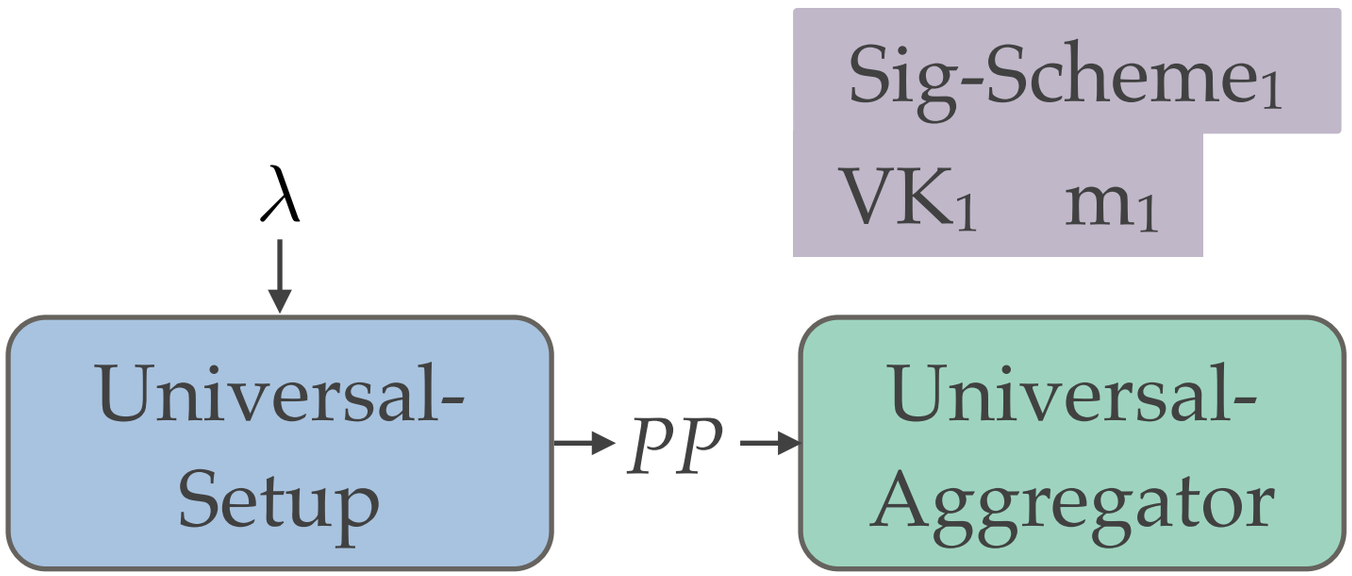
Universal Signature Aggregators



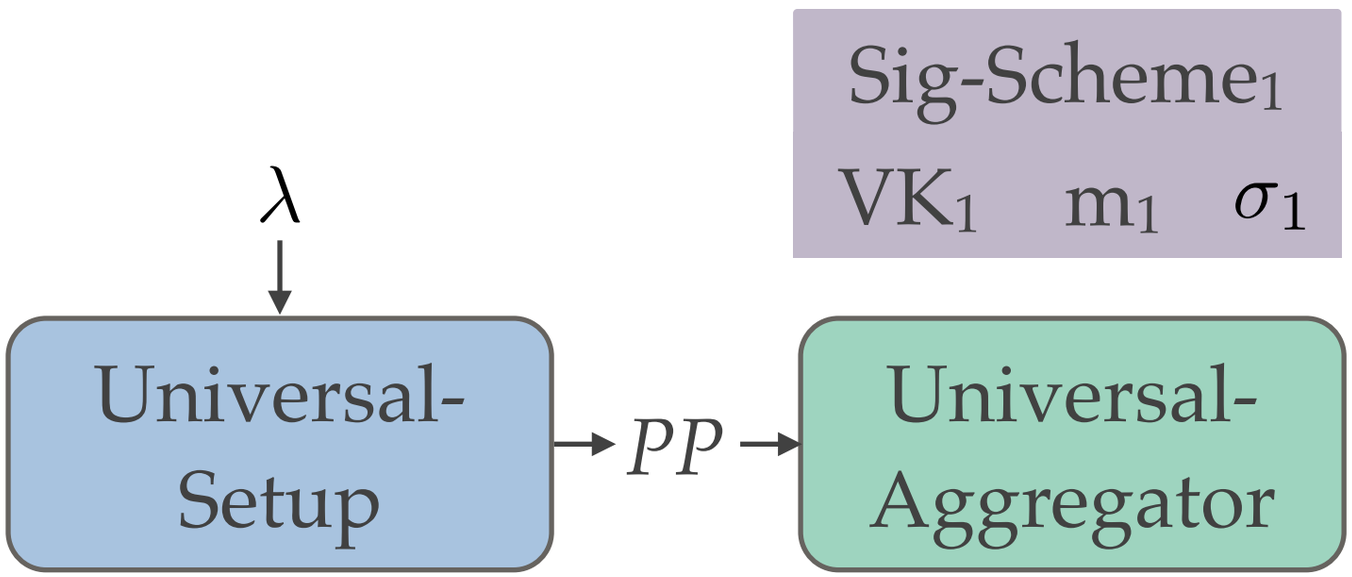
Universal Signature Aggregators



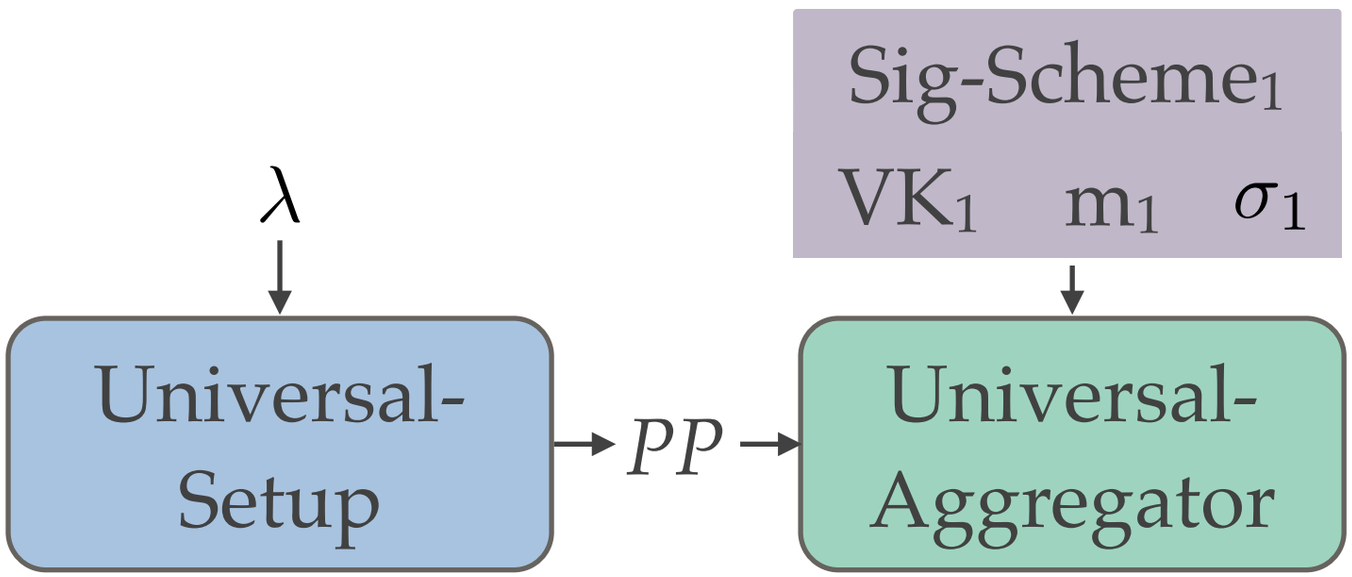
Universal Signature Aggregators



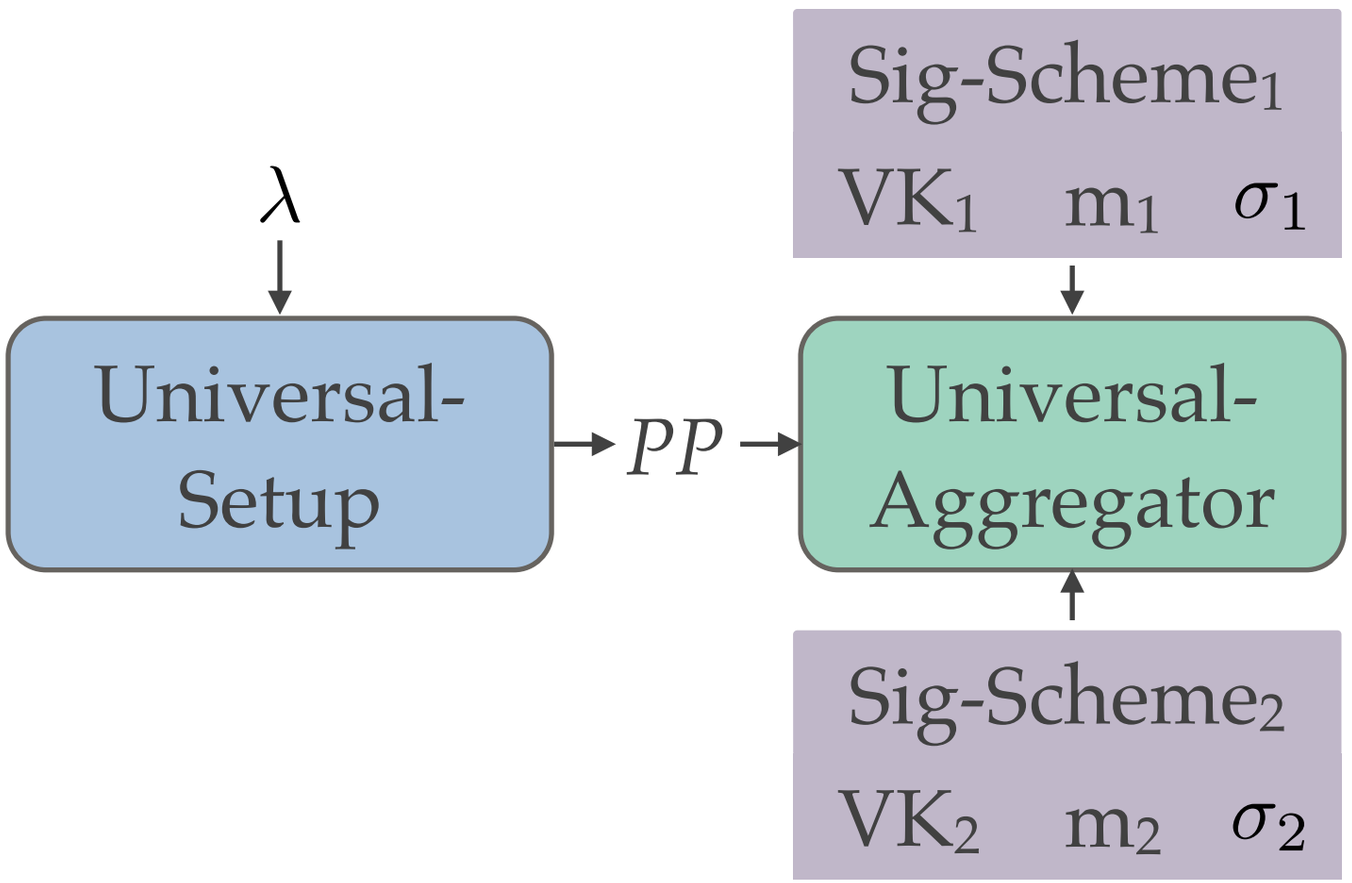
Universal Signature Aggregators



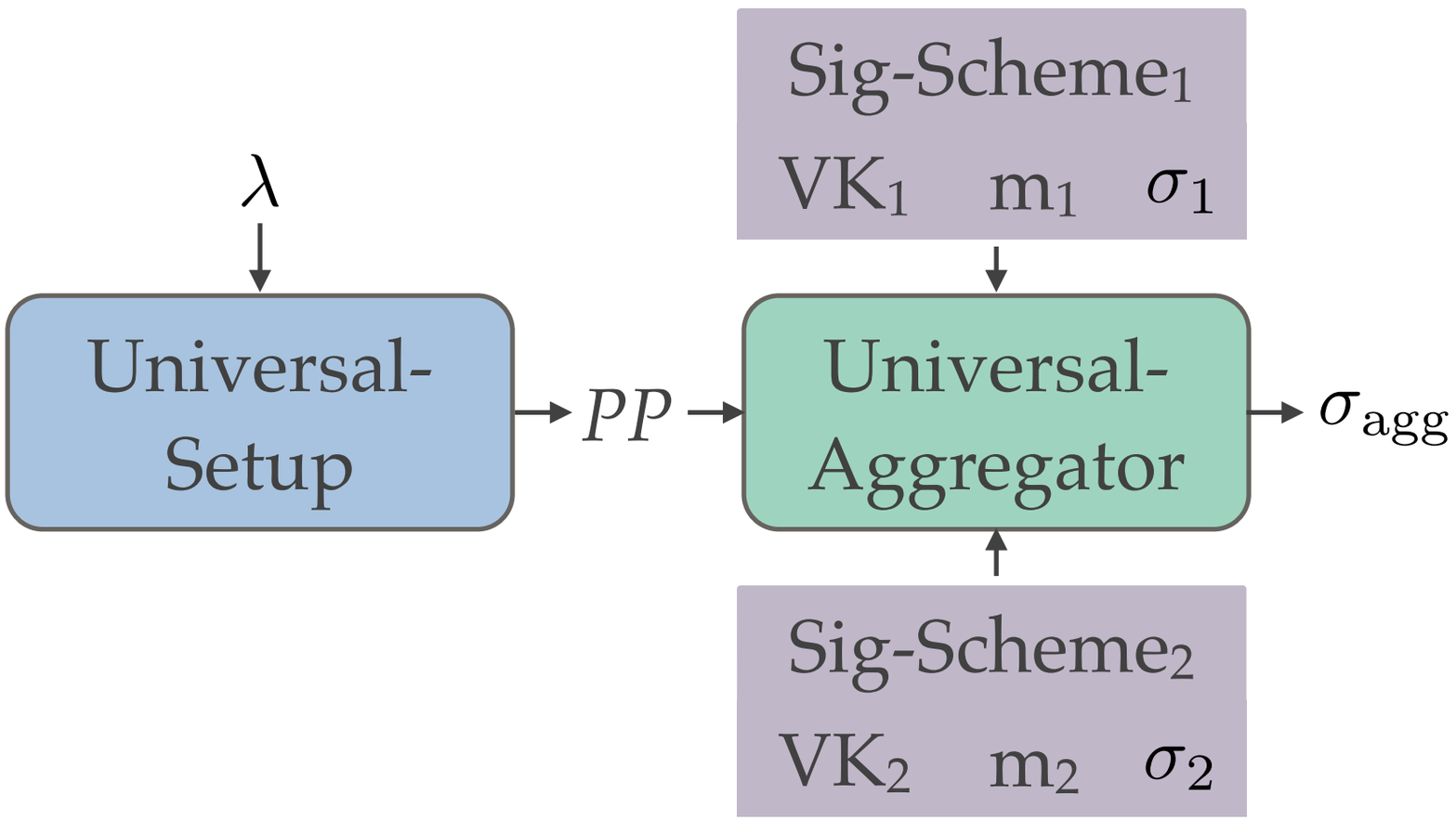
Universal Signature Aggregators



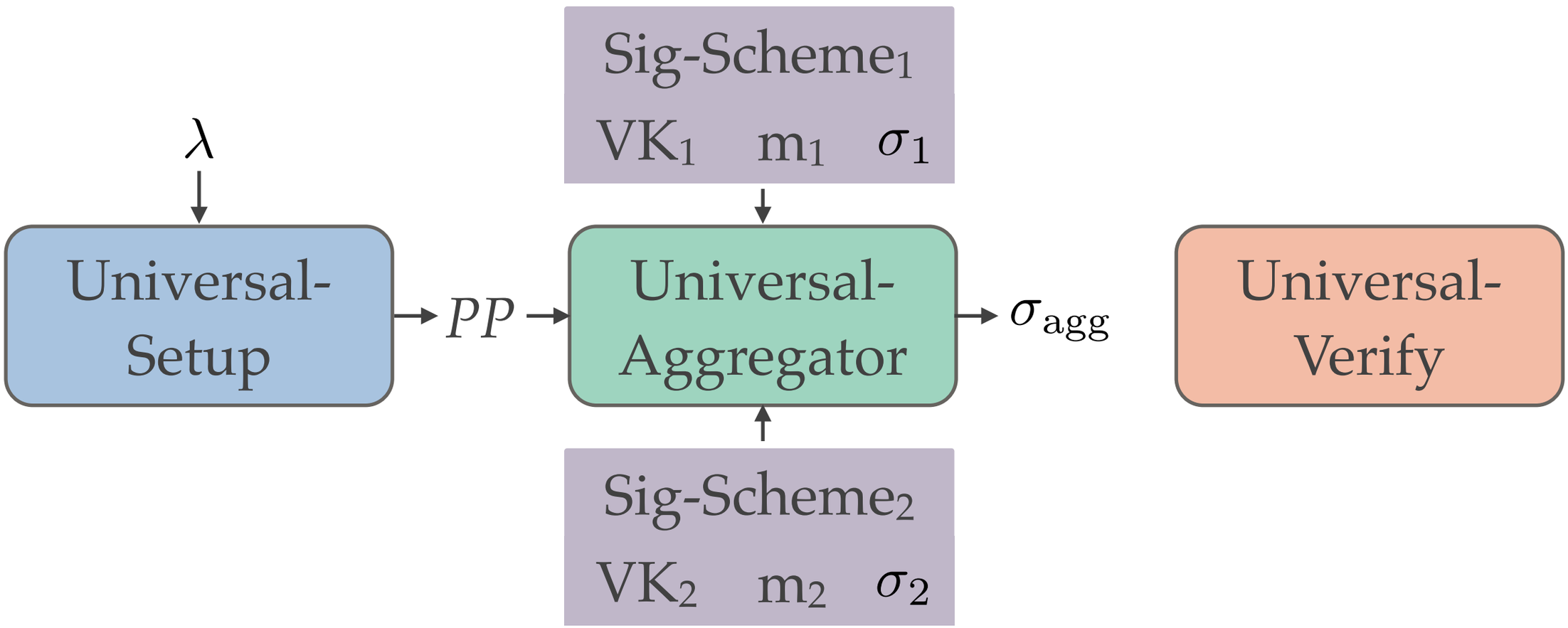
Universal Signature Aggregators



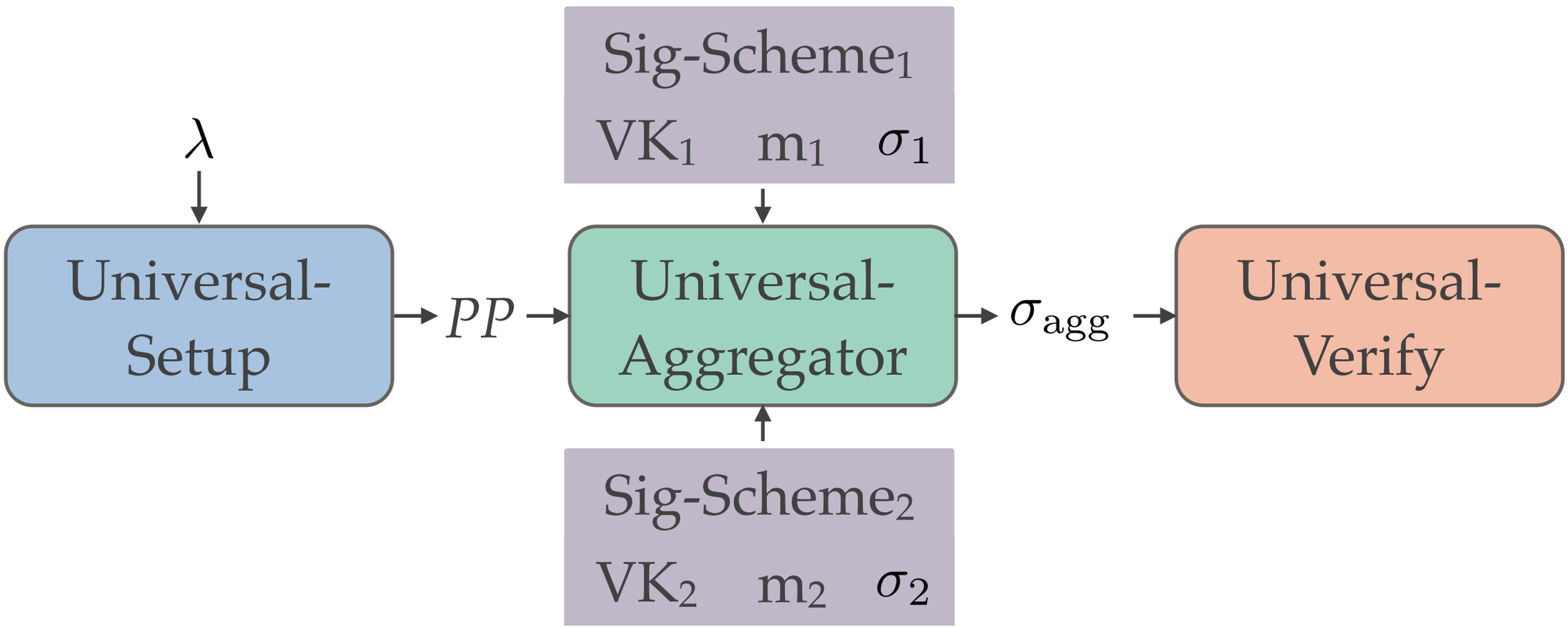
Universal Signature Aggregators



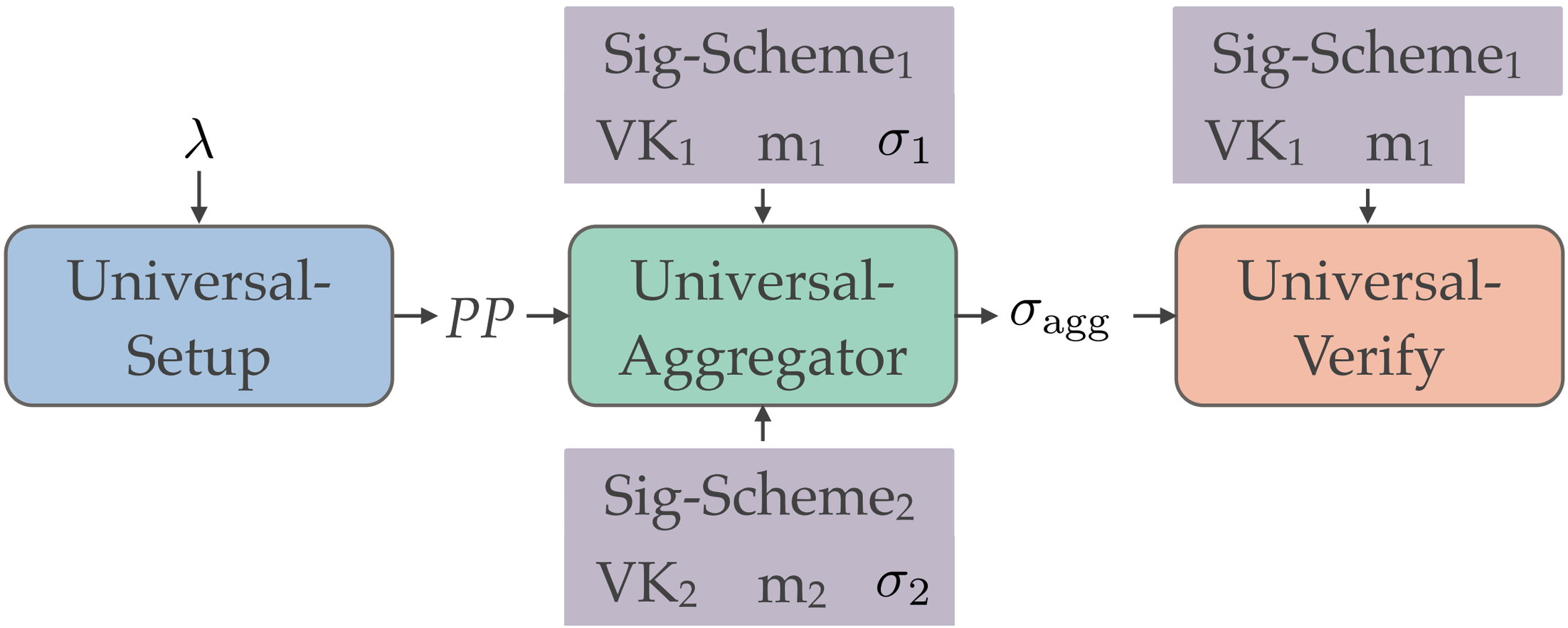
Universal Signature Aggregators



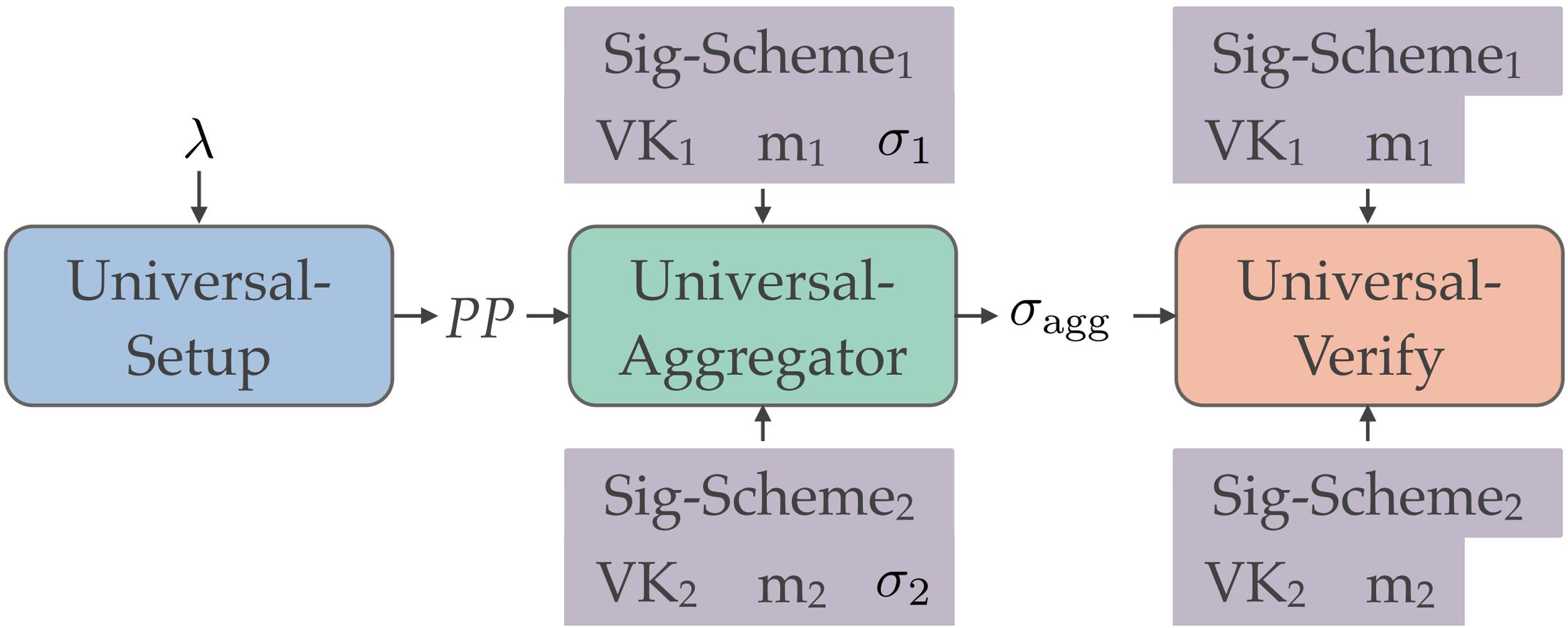
Universal Signature Aggregators



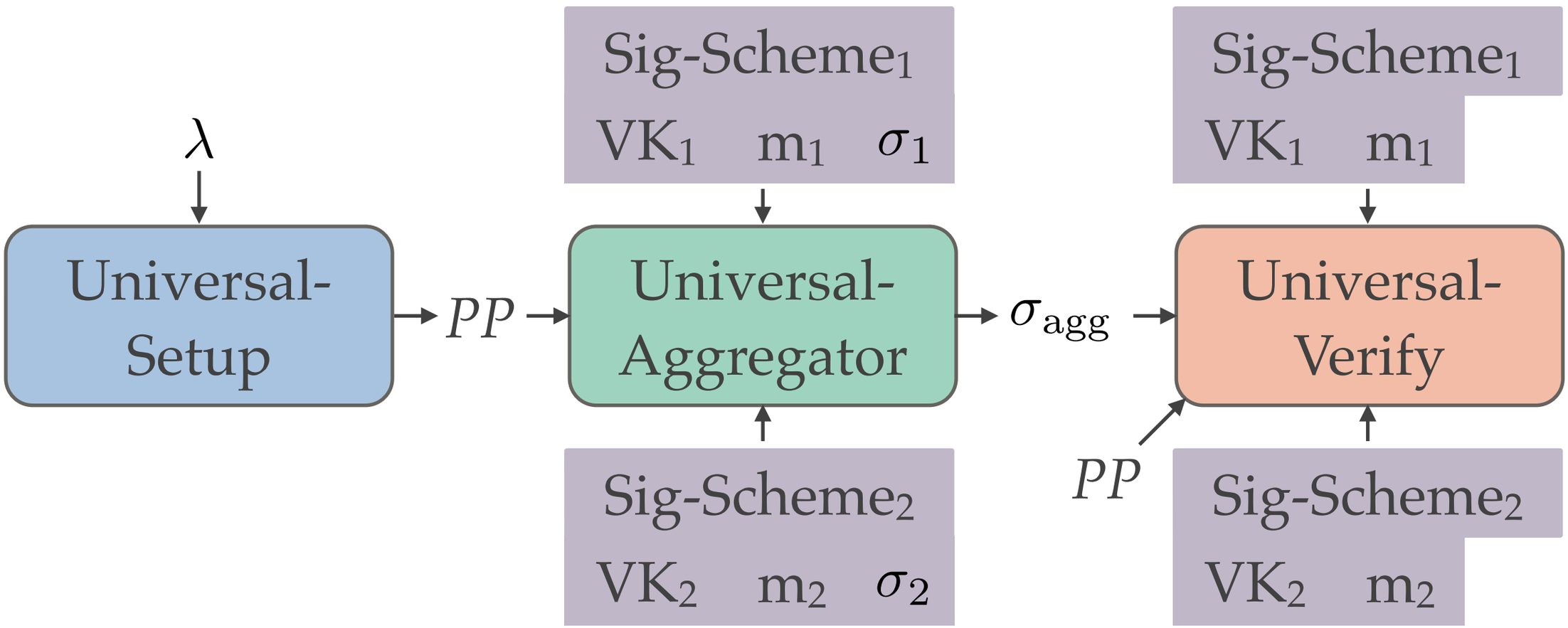
Universal Signature Aggregators



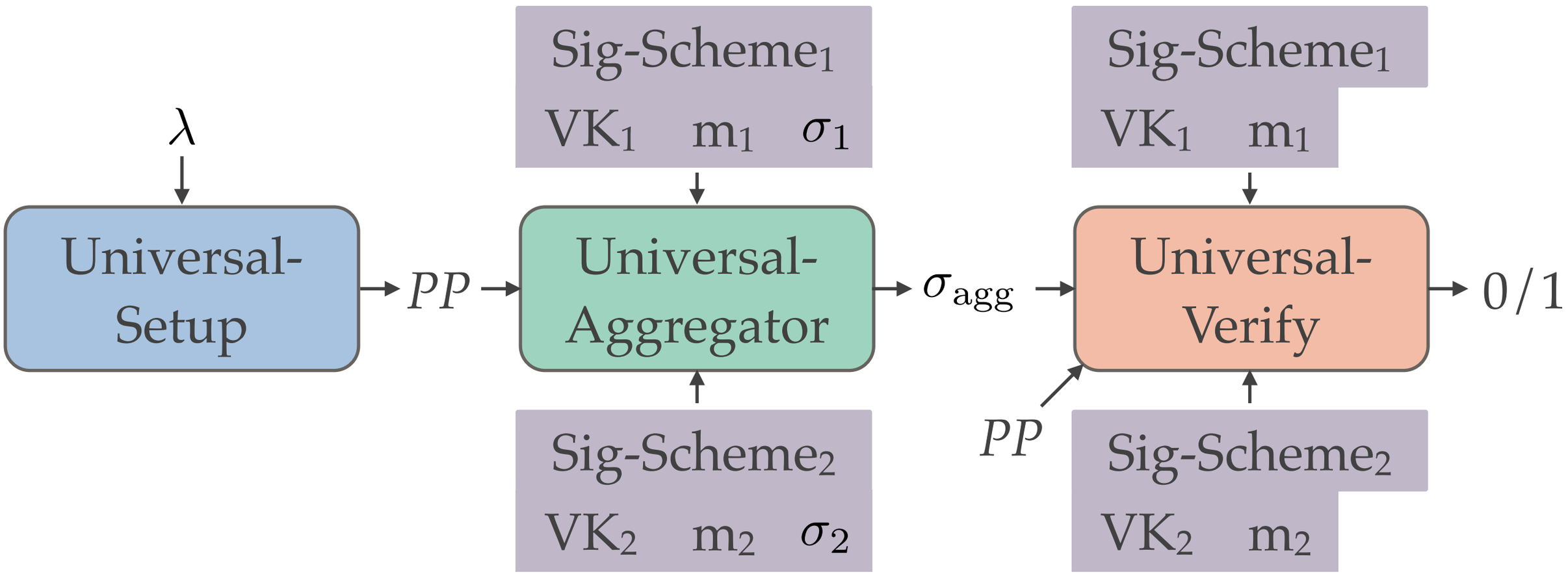
Universal Signature Aggregators



Universal Signature Aggregators



Universal Signature Aggregators



Universal Signature Aggregators

Universal Signature Aggregators

Correctness

Universal Signature Aggregators

Correctness

σ_1, σ_2 are valid signatures for m_1, m_2,

$\Longrightarrow$ Universal-Verify accepts σ_{agg}.

Universal Signature Aggregators

Correctness

σ_1, σ_2 are valid signatures for m_1, m_2,

$\Longrightarrow$ Universal-Verify accepts σ_{agg}.

Succinctness

Universal Signature Aggregators

Correctness

σ_1, σ_2 are valid signatures for m_1, m_2,

$\Longrightarrow$ Universal-Verify accepts σ_{agg}.

Succinctness

Size of σ_{agg} depends only on λ
(not on number of signatures aggregated)

Universal Signature Aggregators

Correctness

σ_1, σ_2 are valid signatures for m_1, m_2,

$\Longrightarrow$ Universal-Verify accepts σ_{agg}.

Succinctness

Size of σ_{agg} depends only on λ
(not on number of signatures aggregated)

Security

Universal Signature Aggregators

Correctness

σ_1, σ_2 are valid signatures for m_1, m_2,

$\Longrightarrow$ Universal-Verify accepts σ_{agg}.

Succinctness

Size of σ_{agg} depends only on λ
(not on number of signatures aggregated)

Security

"If Sig-Scheme$_1$, Sig-Scheme$_2$ are secure,
then it is hard to fool Universal-Verify"

Universal Signature Aggregators: Security

Security

Universal Signature Aggregators: Security

Security

Att-Scheme = (Setup, Sign, Verify-ckt)

Universal Signature Aggregators: Security

Security

Att-Scheme = (Setup, Sign, Verify-ckt)

(SK, VK) ← Setup

Universal Signature Aggregators: Security

Security

Att-Scheme = (Setup, Sign, Verify-ckt)

(SK, VK) ← Setup

PP ← Universal-Setup

Universal Signature Aggregators: Security

Security

Att-Scheme = (Setup, Sign, Verify-ckt)

(SK, VK) ← Setup

PP ← Universal-Setup

PP, VK

Universal Signature Aggregators: Security

Security

Att-Scheme = (Setup, Sign, Verify-ckt)

(SK, VK) ← Setup

PP ← Universal-Setup

PP, VK

m_i

Universal Signature Aggregators: Security

Security

Att-Scheme = (Setup, Sign, Verify-ckt)

(SK, VK) ← Setup

PP ← Universal-Setup

PP, VK

m_i

$\sigma_i = \text{Sign}(m_i, SK)$

Universal Signature Aggregators: Security

Security

Att-Scheme = (Setup, Sign, Verify-ckt)

(SK, VK) ← Setup

PP ← Universal-Setup

PP, VK

m_i

$\sigma_i = \text{Sign}(m_i, SK)$

σ_i

Universal Signature Aggregators: Security

Security

Att-Scheme = (Setup, Sign, Verify-ckt)

(SK, VK) ← Setup

PP ← Universal-Setup

PP, VK

m_i

$\sigma_i = \text{Sign}(m_i, SK)$

σ_i

{ Att-Scheme, m_1^*, VK}

{Scheme$_2$, m_2^*, VK_2^*} … {Scheme$_n$, m_n^*, VK_n^*}

σ_{agg}

Universal Signature Aggregators: Security

Security

Att-Scheme = (Setup, Sign, Verify-ckt)

(SK, VK) ← Setup

PP ← Universal-Setup

PP, VK

m_i

$\sigma_i = \text{Sign}(m_i, SK)$

σ_i

Attacker wins if

{ Att-Scheme, m_1^*, VK}

{Scheme$_2$, m_2^*, VK_2^*} … {Scheme$_n$, m_n^*, VK_n^*}

σ_{agg}

Universal Signature Aggregators: Security

Security

Att-Scheme = (Setup, Sign, Verify-ckt)

(SK, VK) ← Setup

PP ← Universal-Setup

PP, VK

m_i

$\sigma_i = \text{Sign}(m_i, SK)$

σ_i

Attacker wins if
- m_1^* not queried

{ Att-Scheme, m_1^*, VK}

{Scheme$_2$, m_2^*, VK_2^*} … {Scheme$_n$, m_n^*, VK_n^*}

σ_{agg}

Universal Signature Aggregators: Security

Security

Att-Scheme = (Setup, Sign, Verify-ckt)

(SK, VK) ← Setup

PP ← Universal-Setup

PP, VK

m_i

$\sigma_i = \text{Sign}(m_i, SK)$

σ_i

Attacker wins if
- m_1^* not queried
- Universal-Verify accepts

{ Att-Scheme, m_1^*, VK}

{Scheme$_2$, m_2^*, VK_2^*} … {Scheme$_n$, m_n^*, VK_n^*}

σ_{agg}

Universal Signature Aggregators: Security

Selective Security

Att-Scheme = (Setup, Sign, Verify-ckt)

(SK, VK) ← Setup

PP ← Universal-Setup

PP, VK

m_i

$\sigma_i = \text{Sign}(m_i, SK)$

σ_i

Attacker wins if
- m_1^* not queried
- Universal-Verify accepts

{ Att-Scheme, m_1^*, VK}

{Scheme$_2$, m_2^*, VK_2^*} … {Scheme$_n$, m_n^*, VK_n^*}

σ_{agg}

Universal Signature Aggregators: Security

Selective Security

Att-Scheme = (Setup, Sign, Verify-ckt)

m_1^*

(SK, VK) ← Setup

PP ← Universal-Setup

PP, VK

m_i

$\sigma_i = \text{Sign}(m_i, SK)$

σ_i

Attacker wins if
- m_1^* not queried
- Universal-Verify accepts

{ Att-Scheme, m_1^*, VK}

{Scheme$_2$, m_2^*, VK_2^*} … {Scheme$_n$, m_n^*, VK_n^*}

σ_{agg}

Challenge in Proving Security

Challenge in Proving Security

Cannot assume any algebraic structure in the signatures to be aggregated!

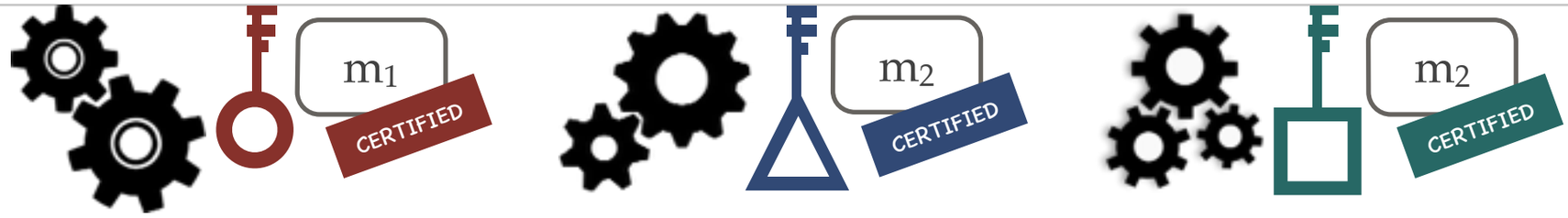
Overview of our Approach

Overview of our Approach

Overview of our Approach

Scheme$_1$ Scheme$_2$

Overview of our Approach



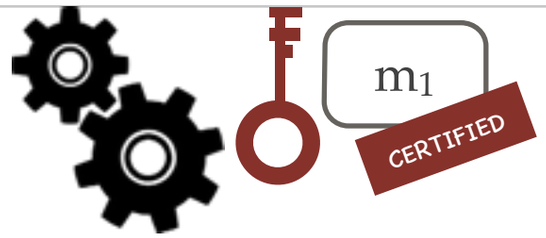
Scheme$_1$ Scheme$_2$ Scheme$_3$

Overview of our Approach

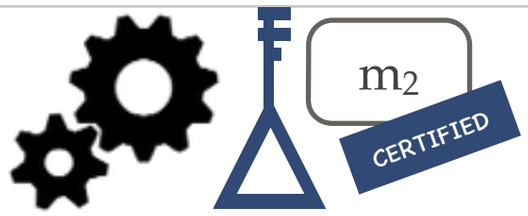
$Scheme_1$ $Scheme_2$ $Scheme_3$

Verify the signatures.

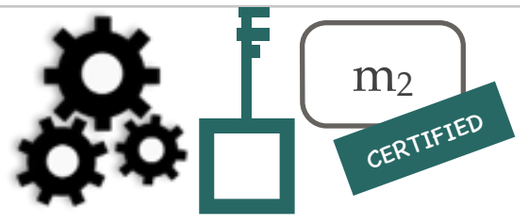
Overview of our Approach



Scheme$_1$

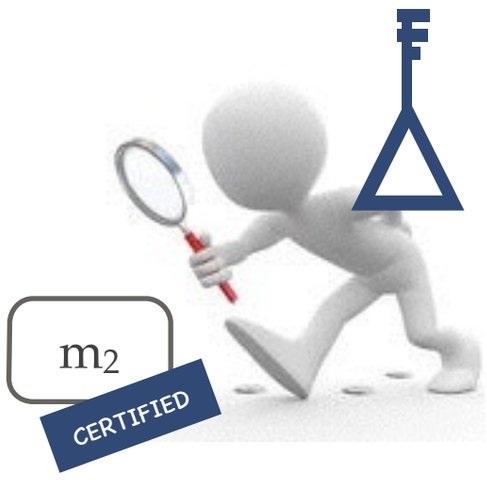


Scheme$_2$

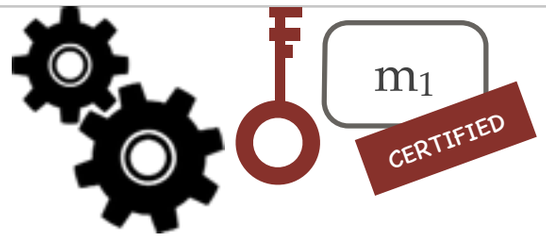


Scheme$_3$

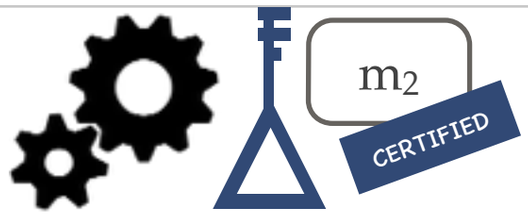
Verify the signatures.



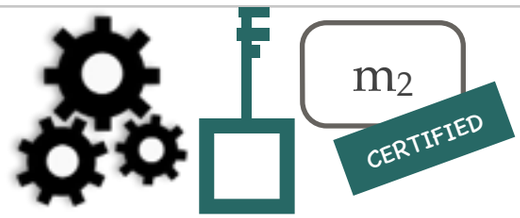
Overview of our Approach



Scheme$_1$



Scheme$_2$

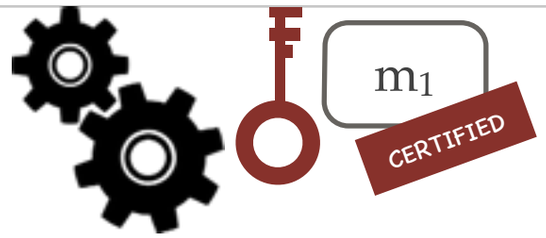


Scheme$_3$

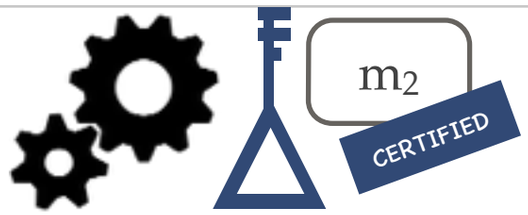
Verify the signatures.

Transform signatures to ‘aggregation-friendly’ form.

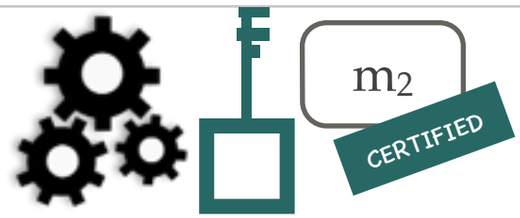
Overview of our Approach



Scheme_1

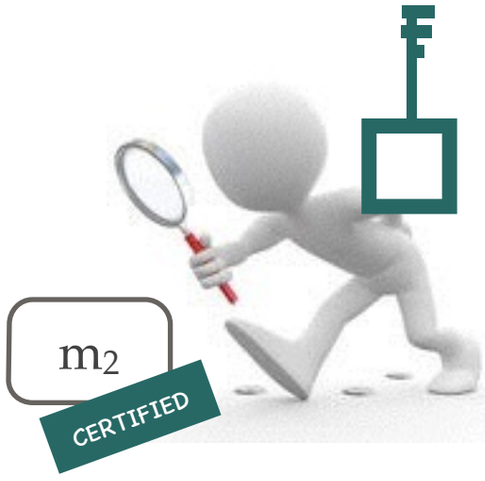


Scheme_2

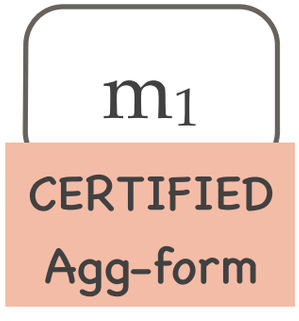


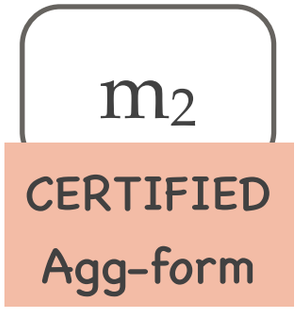
Scheme_3

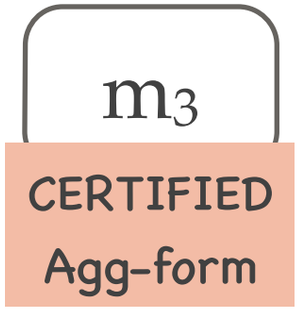
Verify the signatures.



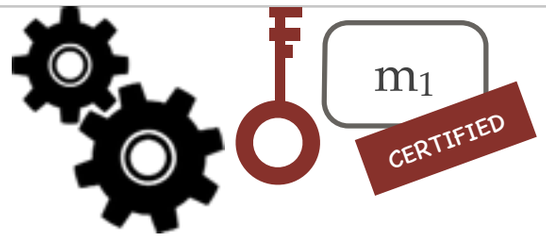
Transform signatures to 'aggregation-friendly' form.



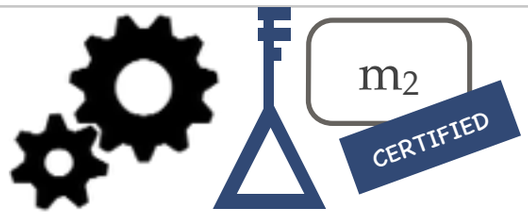




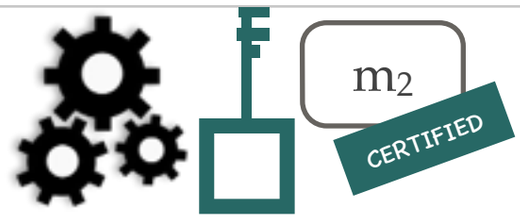
Overview of our Approach



Scheme_1



Scheme_2



Scheme_3

Verify the signatures.

Transform signatures to 'aggregation-friendly' form.

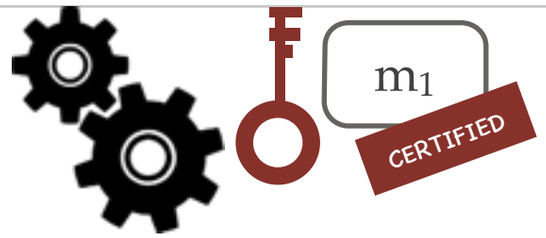
m_1 CERTIFIED Agg-form

m_2 CERTIFIED Agg-form

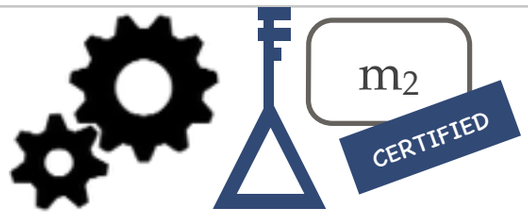
m_3 CERTIFIED Agg-form

Aggregate!

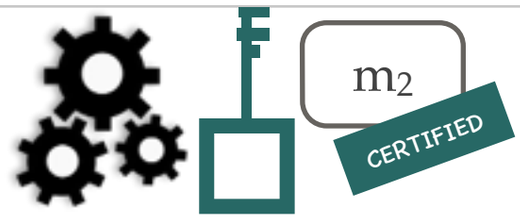
Overview of our Approach



Scheme_1



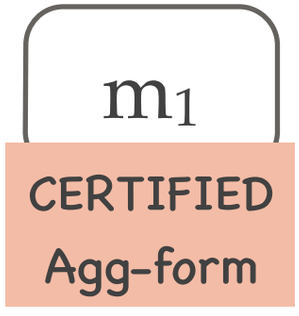
Scheme_2

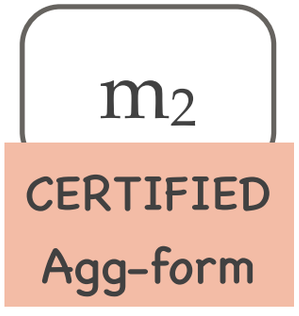


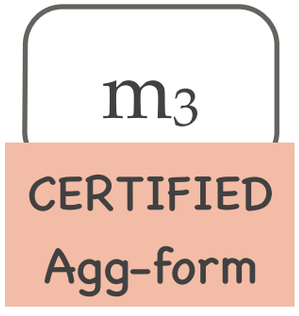
Scheme_3

Verify the signatures.

Transform signatures to 'aggregation-friendly' form.







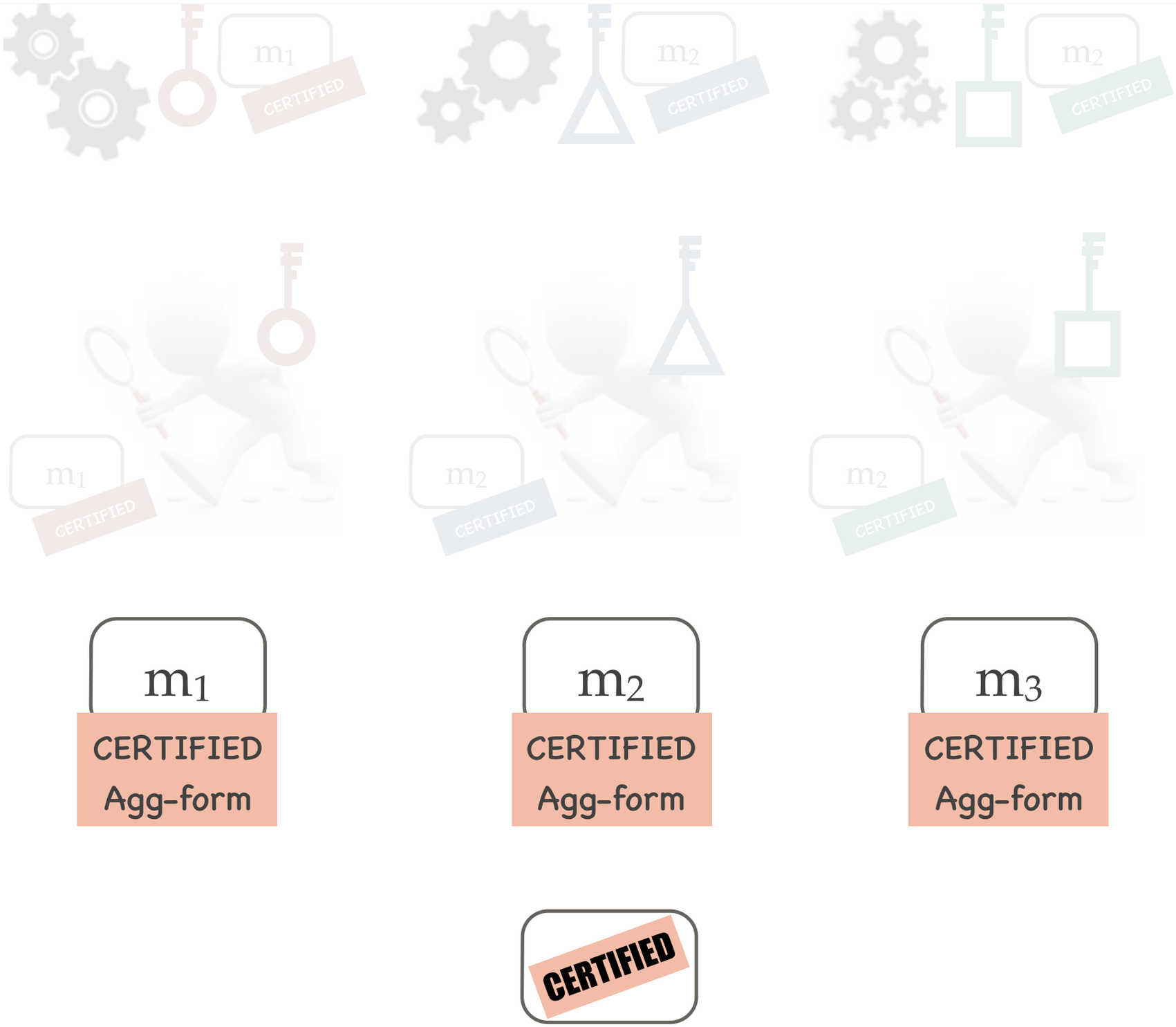
Aggregate!

Overview of our Approach

Verify the signatures.

Transform signatures to ‘aggregation-friendly’ form.

Aggregate!

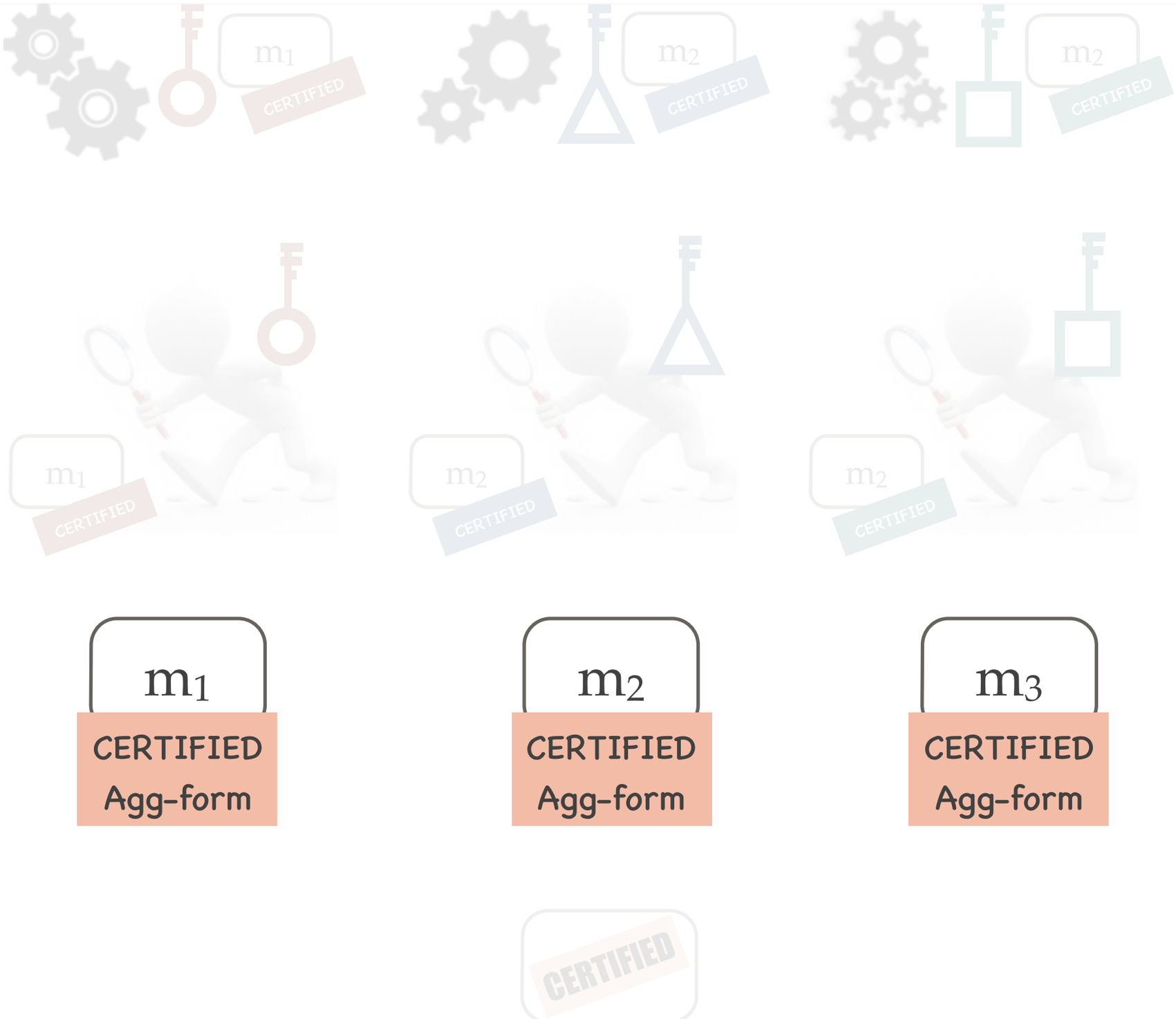


Overview of our Approach

Verify the signatures.

Transform signatures to ‘aggregation-friendly’ form.

Aggregate!



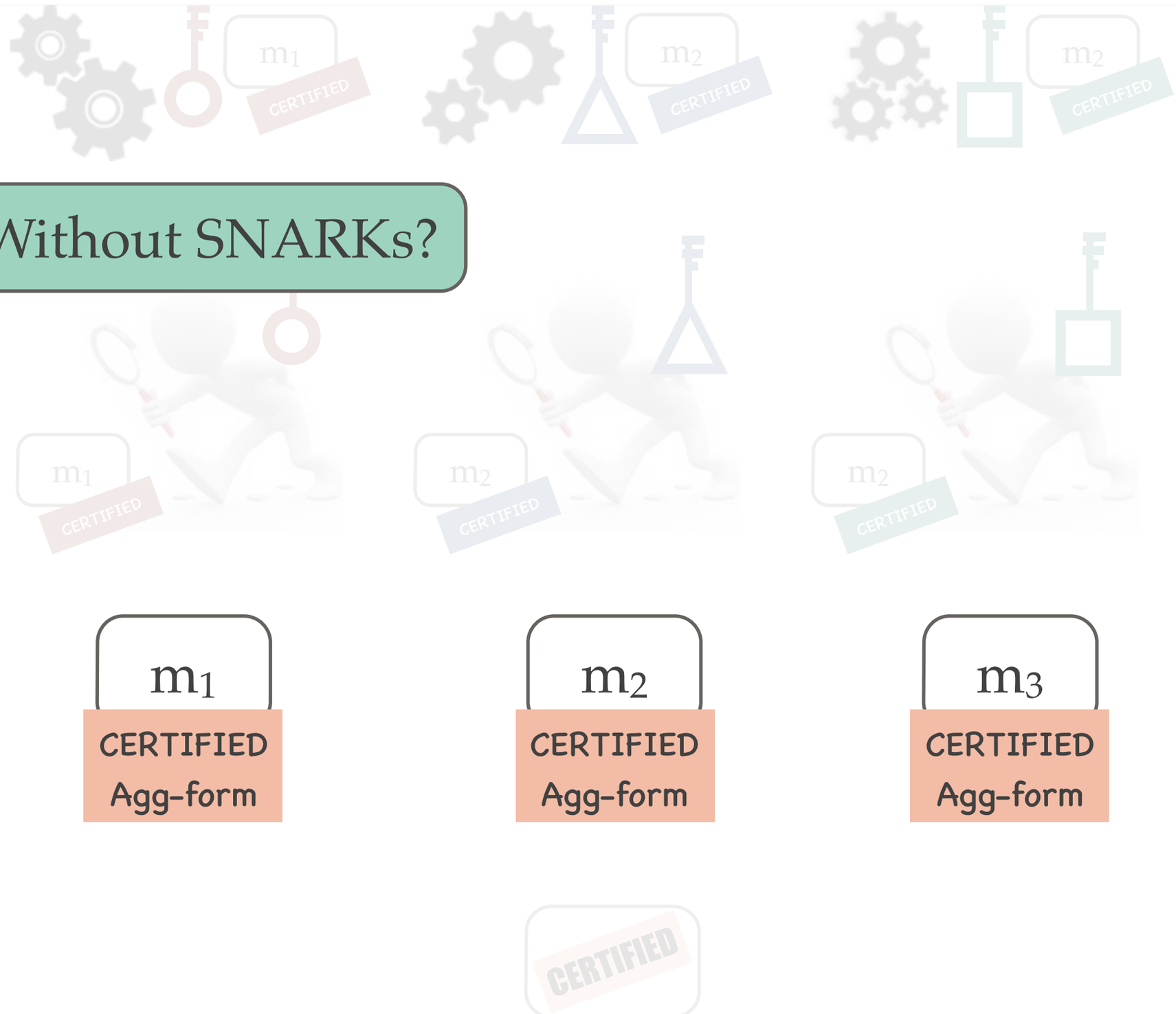
Overview of our Approach

Without SNARKs?

Verify the signatures.

Transform signatures to 'aggregation-friendly' form.

Aggregate!



Overview of our Approach

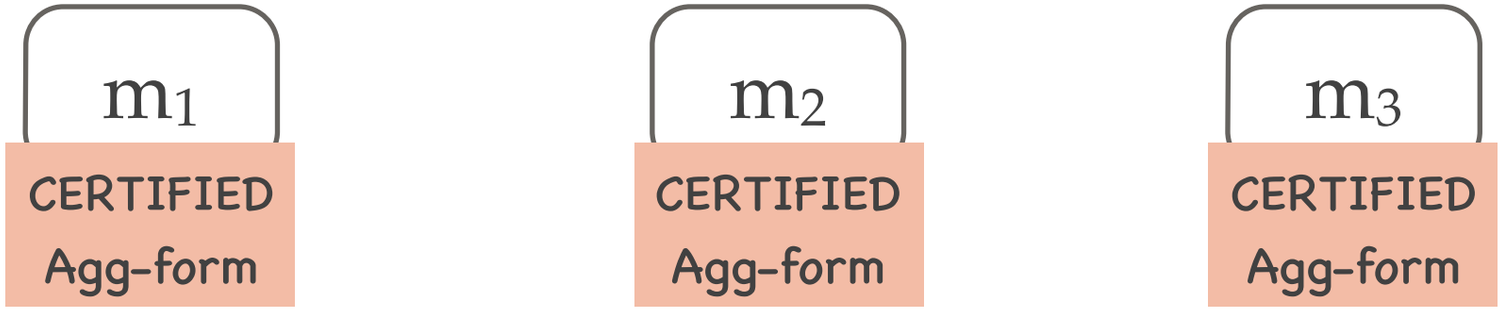
Without SNARKs?

Without knowledge assumptions?

Verify the signatures.

Transform signatures to ‘aggregation-friendly’ form.

Aggregate!



Overview of our Approach

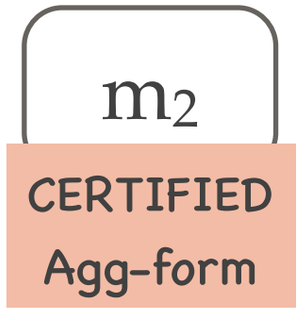
Without SNARKs?

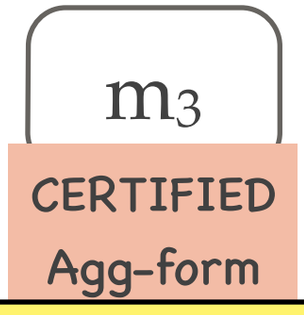
Without knowledge assumptions?

Verify the signatures.

Transform signatures to 'aggregation-friendly' form.

Aggregate!





Code Obfuscation!

Code Obfuscation

Code Obfuscation

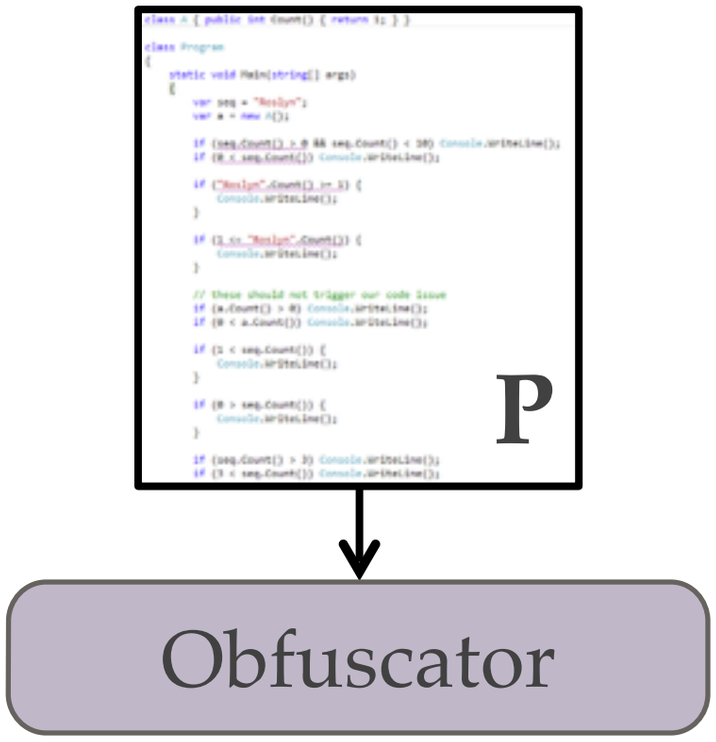
Goal: Make programs maximally unintelligible.

Code Obfuscation

Goal: Make programs maximally unintelligible.

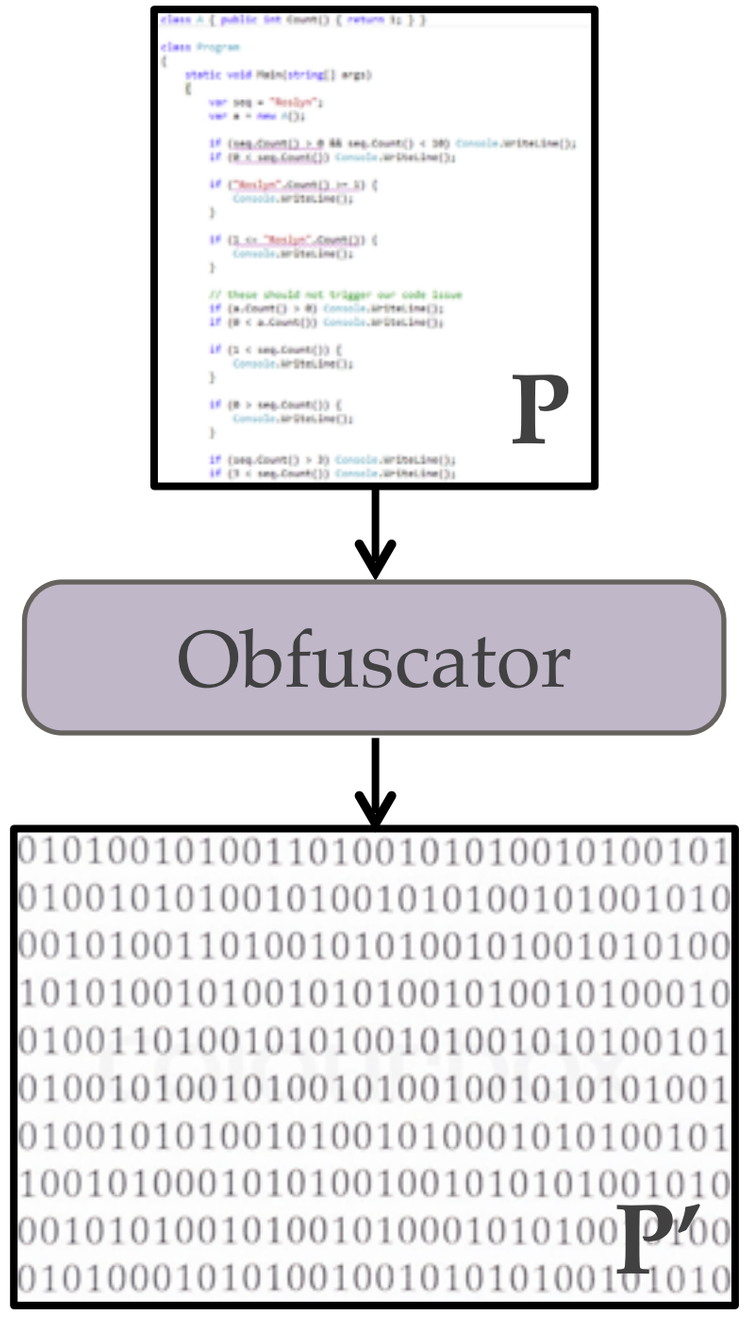
Code Obfuscation

Goal: Make programs maximally unintelligible.



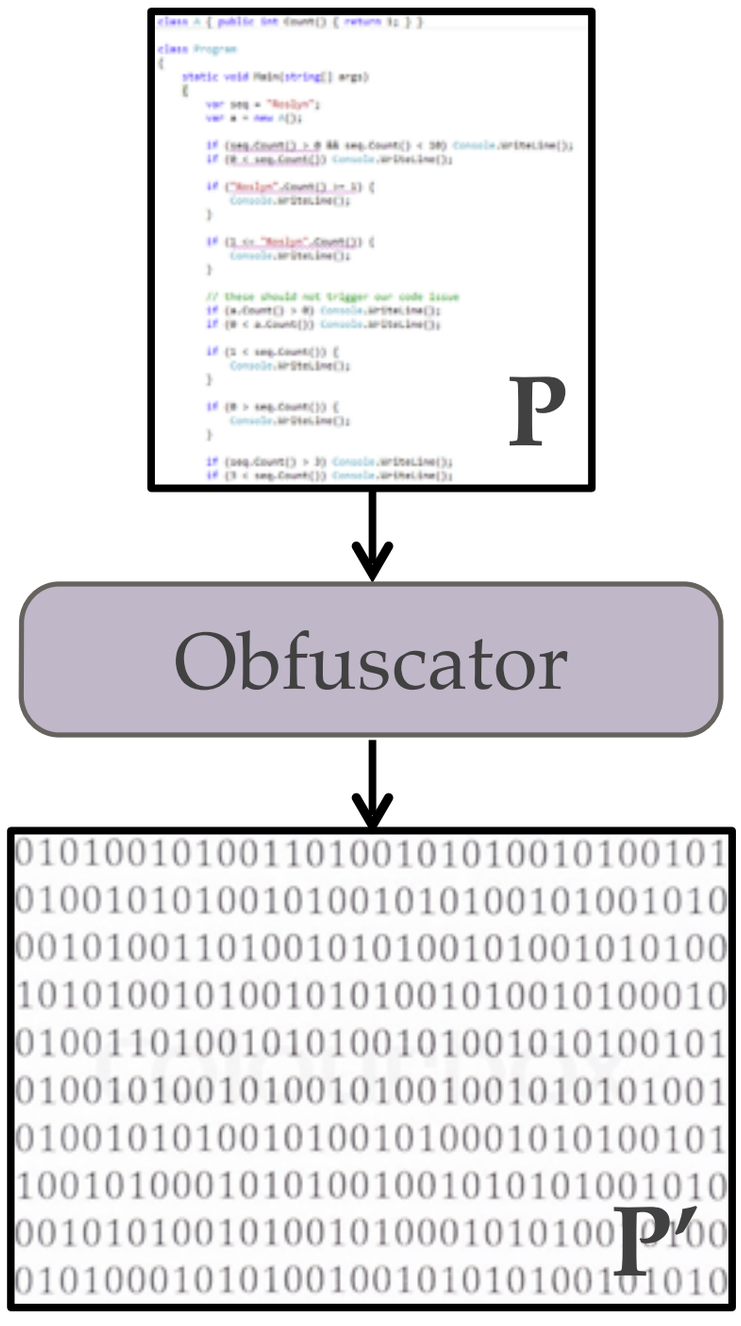
Code Obfuscation

Goal: Make programs maximally unintelligible.



Code Obfuscation

Goal: Make programs maximally unintelligible.



Code Obfuscation

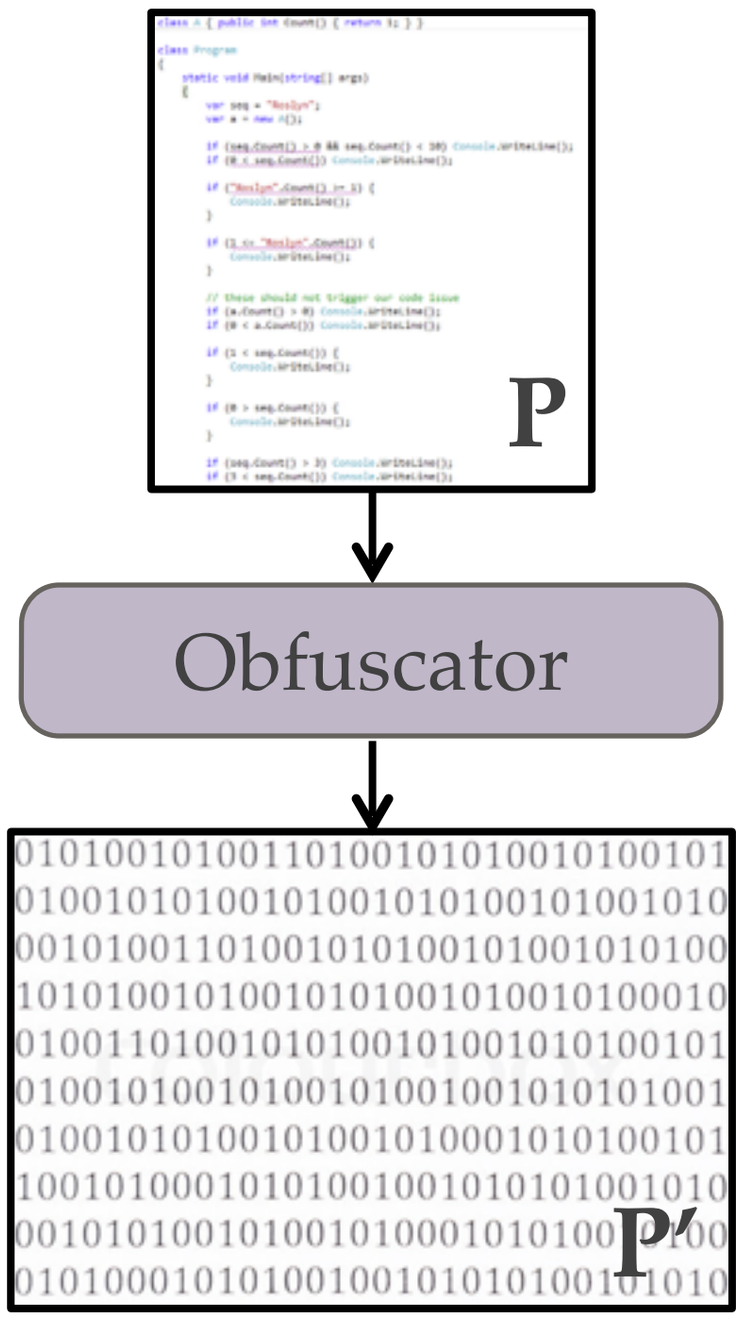
Goal: Make programs maximally unintelligible.

Virtual Black Box Obfuscator

Having obfuscated code

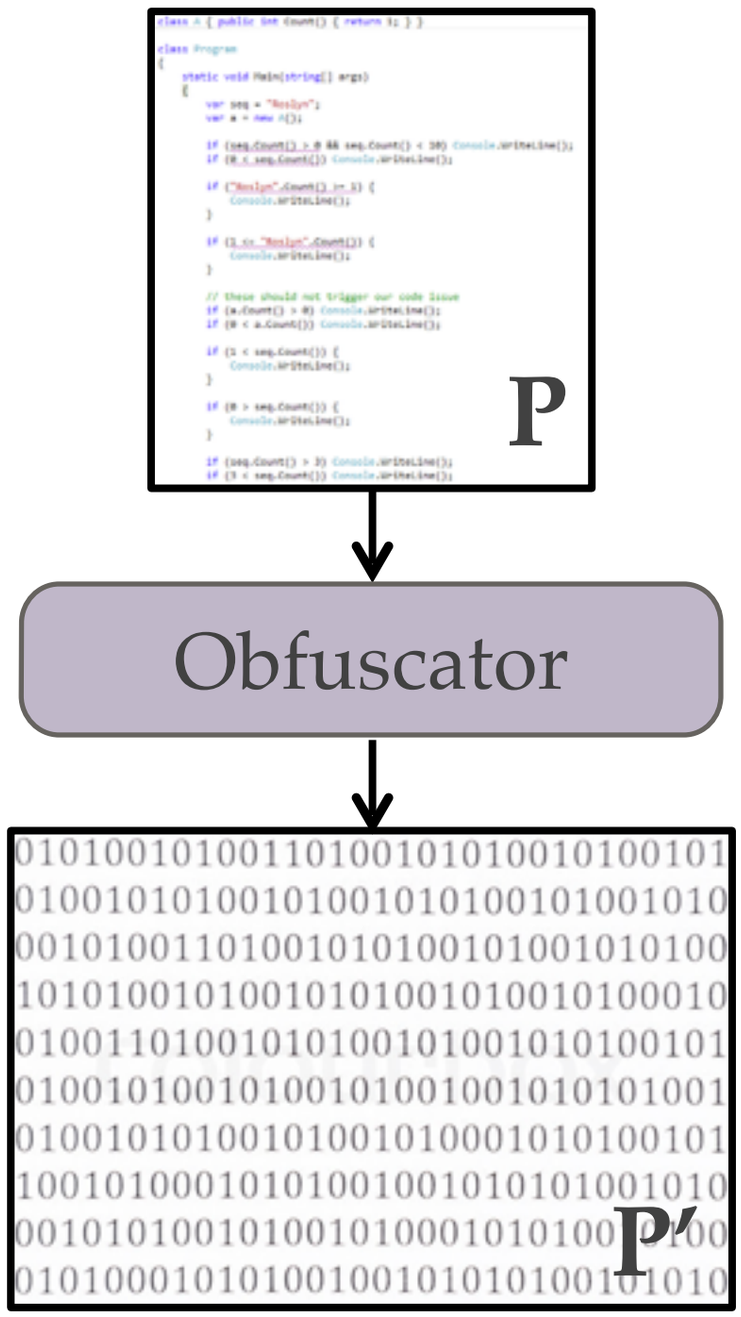
≈

Having black box access to code



Code Obfuscation

Goal: Make programs maximally unintelligible.



Virtual Black Box Obfuscator

Having obfuscated code

≈

Having black box access to code

✘ [BGIRSVY01]

Code Obfuscation

Goal: Make programs maximally unintelligible.

Code Obfuscation

Goal: Make programs maximally unintelligible.

Indistinguishability Obfuscator

C_0, C_1 functionally identical circuits.

$iO(C_0) \approx iO(C_1)$

Code Obfuscation

Goal: Make programs maximally unintelligible.

[BGIRSVY01] negative result does not apply for *iO*.

Indistinguishability Obfuscator

C_0, C_1 functionally identical circuits.

$iO(C_0) \approx iO(C_1)$

Code Obfuscation

Goal: Make programs maximally unintelligible.

Indistinguishability Obfuscator

C_0, C_1 functionally identical circuits.

$iO(C_0) \approx iO(C_1)$

[BGIRSVY01] negative result does not apply for *iO*.

[GGHRSW13] gave a candidate construction for *iO*.

Our Constructions

Our Constructions

iO + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for unique signature schemes

Our Constructions

iO + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

Our Constructions

iO + RSA + OWFs
$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

VBB obfuscation + RSA + OWFs
$\Longrightarrow$ Universal Aggregator for **all** signature schemes

Our Constructions

iO + RSA + OWFs
$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

VBB obfuscation + RSA + OWFs
$\Longrightarrow$ Universal Aggregator for **all** signature schemes

Adaptive

Our Constructions

iO + RSA + OWFs
$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

VBB obfuscation + RSA + OWFs
$\Longrightarrow$ Universal Aggregator for **all** signature schemes

Adaptive

Homomorphic Enc. + iO + OWFs
$\Longrightarrow$ Universal Aggregator for **all** signature schemes

Our Constructions

iO + RSA + OWFs
$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

VBB obfuscation + RSA + OWFs
$\Longrightarrow$ Universal Aggregator for **all** signature schemes

Adaptive

subexp.

Homomorphic Enc. + iO + OWFs
$\Longrightarrow$ Universal Aggregator for **all** signature schemes

Our Constructions

iO + RSA + OWFs
$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

VBB obfuscation + RSA + OWFs
$\Longrightarrow$ Universal Aggregator for **all** signature schemes

Adaptive

subexp.

Homomorphic Enc. + iO + OWFs
$\Longrightarrow$ Universal Aggregator for **all** signature schemes

Selective

Our Constructions

iO + RSA + OWFs
$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

VBB obfuscation + RSA + OWFs
$\Longrightarrow$ Universal Aggregator for **all** signature schemes

Adaptive

subexp.

Homomorphic Enc. + iO + OWFs
$\Longrightarrow$ Universal Aggregator for **all** signature schemes

Selective

Construction 1

Construction 1

iO + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

Construction 1

iO + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

Construction 1

iO + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

Unique Signature Scheme: For every m, VK, there exists exactly one signature that verifies.

Construction 1

iO + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

Unique Signature Scheme: For every m, VK, there exists exactly one signature that verifies.

eg. RSA based Full Domain Hash signatures (BR-93)

Construction 1

iO + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

Construction 1

iO + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

Ingredients

Construction 1

iO + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

Ingredients

iO

Construction 1

iO + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

Ingredients

iO

RSA: modulus N

Construction 1

iO + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for unique signature schemes

Selective

Ingredients

iO

RSA: modulus N

(puncturable) PRF F : {0,1}*$\rightarrow \mathbb{Z}_N^*$

Construction 1

Construction 1

Universal-Setup

Construction 1

Universal-Setup

1.Choose RSA modulus N, exponent e, PRF key K.

Construction 1

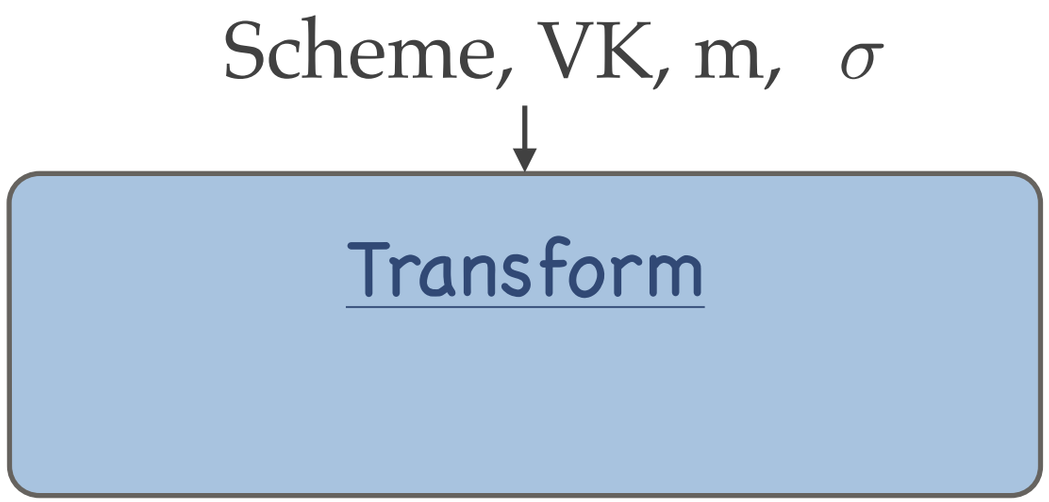
Universal-Setup

1. Choose RSA modulus N, exponent e, PRF key K.
2. Output PP = (iO(Transform), iO(Transform-Verify), N, e)

Construction 1

Universal-Setup

1. Choose RSA modulus N, exponent e, PRF key K.
2. Output PP = (iO(Transform), iO(Transform-Verify), N, e)



Construction 1

Universal-Setup

1.Choose RSA modulus N, exponent e, PRF key K.

2.Output PP = (iO(Transform), iO(Transform-Verify), N, e)

Scheme, VK, m, σ

↓

Transform

1.Check Verify-ckt(VK, m, σ) = 1

Construction 1

Universal-Setup

1. Choose RSA modulus N, exponent e, PRF key K.
2. Output PP = (iO(Transform), iO(Transform-Verify), N, e)

Scheme, VK, m, σ

↓

Transform

1. Check Verify-ckt(VK, m, σ) = 1
2. Output F(K, (Scheme, VK, m))

Construction 1

Universal-Setup

1. Choose RSA modulus N, exponent e, PRF key K.
2. Output PP = (iO(Transform), iO(Transform-Verify), N, e)

Scheme, VK, m, σ

↓

Transform

1. Check Verify-ckt(VK, m, σ) = 1
2. Output F(K, (Scheme, VK, m))

$\in \mathbb{Z}_N^*$

Construction 1

Universal-Setup

1. Choose RSA modulus N, exponent e, PRF key K.
2. Output PP = (iO(Transform), iO(Transform-Verify), N, e)

Scheme, VK, m, σ

↓

Transform

1. Check Verify-ckt(VK, m, σ) = 1
2. Output F(K, (Scheme, VK, m))

$\in \mathbb{Z}_N^*$

Scheme, VK, m

↓

Transform-Verify

Construction 1

Universal-Setup

1. Choose RSA modulus N, exponent e, PRF key K.
2. Output PP = (iO(Transform), iO(Transform-Verify), N, e)

Scheme, VK, m, σ

Transform

1. Check Verify-ckt(VK, m, σ) = 1
2. Output F(K, (Scheme, VK, m))

$\in \mathbb{Z}_N^*$

Scheme, VK, m

Transform-Verify

1. Output $F(K, (\text{Scheme}, VK, m))^e$

Construction 1

Universal-Setup

1. Choose RSA modulus N, exponent e, PRF key K.
2. Output PP = (iO(Transform), iO(Transform-Verify), N, e)

Scheme, VK, m, σ

↓

Transform

1. Check Verify-ckt(VK, m, σ) = 1
2. Output F(K, (Scheme, VK, m))

$\in \mathbb{Z}_N^*$

Scheme, VK, m

↓

Transform-Verify

1. Output F(K, (Scheme, VK, m))^e

$\in \mathbb{Z}_N^*$

Construction 1

Construction 1

Universal-Aggregator

Construction 1

Universal-Aggregator

Inputs: PP = (P_1, P_2, N, e), {($Scheme_i$, VK_i, m_i, σ_i)}

Construction 1

Universal-Aggregator

Inputs: PP = (P_1, P_2, N, e), {(Scheme$_i$, VK$_i$, m$_i$, σ_i)}

Scheme, VK, m, σ

↓

Transform

1.Check Verify-ckt(VK, m, σ) = 1

2.Output F(K, (Scheme, VK, m))

Construction 1

Universal-Aggregator

Inputs: $PP = (P_1, P_2, N, e)$, $\{(Scheme_i, VK_i, m_i, \sigma_i)\}$

1. Compute $t_i = P_1(Scheme_i, VK_i, m_i, \sigma_i)$

Scheme, VK, m, σ

↓

Transform

1.Check Verify-ckt(VK, m, σ) = 1

2.Output F(K, (Scheme, VK, m))

Construction 1

Universal-Aggregator

Inputs: $PP = (P_1, P_2, N, e)$, $\{(Scheme_i, VK_i, m_i, \sigma_i)\}$

1. Compute $t_i = P_1(Scheme_i, VK_i, m_i, \sigma_i)$ $\in \mathbb{Z}_N^*$

Scheme, VK, m, σ

↓

Transform

1. Check Verify-ckt(VK, m, σ) = 1
2. Output F(K, (Scheme, VK, m))

Construction 1

Universal-Aggregator

Inputs: $PP = (P_1, P_2, N, e)$, $\{(Scheme_i, VK_i, m_i, \sigma_i)\}$

1. Compute $t_i = P_1(Scheme_i, VK_i, m_i, \sigma_i)$ $\in \mathbb{Z}_N^*$
2. Output $\prod t_i$

Scheme, VK, m, σ

↓

Transform

1. Check Verify-ckt(VK, m, σ) = 1
2. Output F(K, (Scheme, VK, m))

Construction 1

Construction 1

Universal-Verify

Construction 1

Universal-Verify

Inputs: $PP = (P_1, P_2, N, e)$, $\{Scheme_i, VK_i, m_i\}$, σ_{agg}

Construction 1

Universal-Verify

Inputs: PP = (P_1, P_2, N, e), {$Scheme_i$, VK_i, m_i}, σ_{agg}

Scheme, VK, m

↓

Transform-Verify

1.Output F(K, (Scheme, VK, m))^e

Construction 1

Universal-Verify

Inputs: $PP = (P_1, P_2, N, e)$, $\{Scheme_i, VK_i, m_i\}$, σ_{agg}

1. Compute $s_i = P_2(Scheme_i, VK_i, m_i)$

Scheme, VK, m

↓

Transform-Verify

1.Output $F(K, (Scheme, VK, m))^e$

Construction 1

Universal-Verify

Inputs: PP = (P_1, P_2, N, e), {$Scheme_i$, VK_i, m_i}, σ_{agg}

1. Compute $s_i = P_2(Scheme_i, VK_i, m_i)$ $\in \mathbb{Z}_N^*$

Scheme, VK, m

↓

Transform-Verify

1.Output $F(K, (Scheme, VK, m))^e$

Construction 1

Universal-Verify

Inputs: $PP = (P_1, P_2, N, e)$, $\{Scheme_i, VK_i, m_i\}$, σ_{agg}

1. Compute $s_i = P_2(Scheme_i, VK_i, m_i)$ ← $\in \mathbb{Z}_N^*$
2. Check $\prod s_i = \sigma_{agg}^e$

Scheme, VK, m

↓

Transform-Verify

1.Output $F(K, (Scheme, VK, m))^e$

Security Proof Idea

Selective proof of security

Security Proof Idea

Selective proof of security

Transform

1. Check Verify-ckt(VK, m, σ) = 1
2. Output F(K, (Scheme, VK, m))

Security Proof Idea

Selective proof of security

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Security Proof Idea

Selective proof of security

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Security Proof Idea

Selective proof of security

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Challenge-tuple fixes unique σ^*.

Security Proof Idea

Security Proof Idea

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Security Proof Idea

30% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Security Proof Idea

30% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

10% win

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Security Proof Idea

30% win

10% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Differ on (challenge tuple, σ^*)

Security Proof Idea

30% win

10% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Differ on (challenge tuple, σ^*)

Use  to extract σ^*

Construction 2

Construction 2

VBB obfuscation + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for **all** signature schemes

Adaptive

Construction 2

VBB obfuscation + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for **all** signature schemes

Adaptive

Construction 1 +

Construction 2

VBB obfuscation + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for **all** signature schemes

Adaptive

Construction 1 +

"Oracle Assimilation"

Construction 2

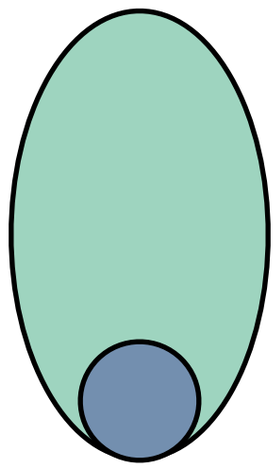
VBB obfuscation + RSA + OWFs

$\Longrightarrow$ Universal Aggregator for **all** signature schemes

Adaptive

Construction 1 +

+



"Oracle Assimilation"

"Partitioning Strategy"

Construction 3

Construction 3

subexp.

Homomorphic Enc. + iO + OWFs

$\Longrightarrow$ Universal Aggregator* for **all** signature schemes

Construction 3

subexp.

Homomorphic Enc. + iO + OWFs

$\Longrightarrow$ Universal Aggregator* for **all** signature schemes

* : Can aggregate only a bounded number of signatures

Construction 3

subexp.

Homomorphic Enc. + iO + OWFs

$\Longrightarrow$ Universal Aggregator* for **all** signature schemes

Ingredients

* : Can aggregate only a bounded number of signatures

Construction 3

subexp.

Homomorphic Enc. + iO + OWFs

$\Longrightarrow$ Universal Aggregator* for **all** signature schemes

Ingredients

iO

* : Can aggregate only a bounded number of signatures

Construction 3

subexp.

Homomorphic Enc. + iO + OWFs

$\Longrightarrow$ Universal Aggregator* for **all** signature schemes

Ingredients

iO

Homomorphic Encryption

* : Can aggregate only a bounded number of signatures

Construction 3

subexp.

Homomorphic Enc. + iO + OWFs

$\Longrightarrow$ Universal Aggregator* for **all** signature schemes

Ingredients

iO

Homomorphic Encryption

(puncturable) PRFs : $\{0,1\}^* \rightarrow \{0,1\}^*$

* : Can aggregate only a bounded number of signatures

Construction 3

subexp.

Homomorphic Enc. + iO + OWFs

$\Longrightarrow$ Universal Aggregator* for **all** signature schemes

Selective

Ingredients

iO

Homomorphic Encryption

(puncturable) PRFs : $\{0,1\}^* \rightarrow \{0,1\}^*$

* : Can aggregate only a bounded number of signatures

Thank you!

Questions?

Security Proof Idea

Security Proof Idea

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Security Proof Idea

30% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Security Proof Idea

30% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

10% win

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Security Proof Idea

30% win

10% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Differ on (challenge tuple, σ^*)

Security Proof Idea

30% win

10% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Differ on (challenge tuple, σ^*)

Use  to extract σ^*

Security Proof Idea

30% win

10% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Differ on (challenge tuple, σ^*)

Security Proof Idea

30% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

10% win

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Differ on (challenge tuple, σ^*)

Transform-Intermediate

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple and $\sigma[1] = 0$, output $\perp$
3.Output F(K, (Scheme, VK, m))

Security Proof Idea

30% win

10% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Identical if $\sigma^*[1] = 1$

Differ on (challenge tuple, σ^*)

Transform-Intermediate

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple and $\sigma[1] = 0$, output $\perp$
3.Output F(K, (Scheme, VK, m))

Security Proof Idea

30% win

10% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Differ on (challenge tuple, σ^*)

Identical if $\sigma^*[1] = 0$

Transform-Intermediate

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple and $\sigma[1] = 0$, output $\perp$
3.Output F(K, (Scheme, VK, m))

Security Proof Idea

30% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

10% win

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Differ on (challenge tuple, σ^*)

Transform-Intermediate

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple and $\sigma[1] = 0$, output $\perp$
3.Output F(K, (Scheme, VK, m))

Security Proof Idea

30% win

10% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Differ on (challenge tuple, σ^*)

Transform-Intermediate

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple and $\sigma[1] = 0$, output $\perp$
3.Output F(K, (Scheme, VK, m))

 ~30% win

Security Proof Idea

30% win

10% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Differ on (challenge tuple, σ^*)

Transform-Intermediate

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple and $\sigma[1] = 0$, output $\perp$
3.Output F(K, (Scheme, VK, m))

~30% win
$\Longrightarrow \sigma^*[1] = 1$

Security Proof Idea

30% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

10% win

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Differ on (challenge tuple, σ^*)

Transform-Intermediate

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple and $\sigma[1] = 0$, output $\perp$
3.Output F(K, (Scheme, VK, m))

Security Proof Idea

30% win

10% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Differ on (challenge tuple, σ^*)

Transform-Intermediate

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple and $\sigma[1] = 0$, output $\perp$
3.Output F(K, (Scheme, VK, m))

 ~10% win

Security Proof Idea

30% win

10% win

Transform

1.Check Verify-ckt(VK, m, σ) = 1
2.Output F(K, (Scheme, VK, m))

Transform'

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple, output $\perp$
3.Output F(K, (Scheme, VK, m))

Differ on (challenge tuple, σ^*)

Transform-Intermediate

1.Check Verify-ckt(VK, m, σ) = 1
2.If (Scheme, VK, m) = challenge-tuple and $\sigma[1] = 0$, output $\perp$
3.Output F(K, (Scheme, VK, m))

~10% win
$\Longrightarrow \sigma^*[1] = 0$