Homomorphic Signatures for Polynomial Functions

Dan Boneh and **David Mandell Freeman**

Stanford University, USA

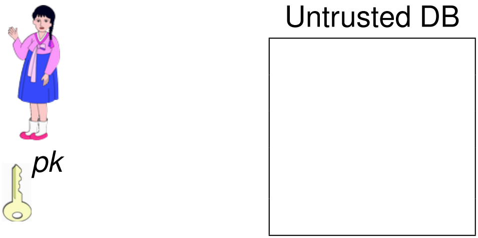
Séminaire de Crypto de l'ENS
4 March 2011

Homomorphic Encryption

Homomorphic encryption allows users to delegate computation while ensuring *secrecy*.

Homomorphic Encryption

Homomorphic encryption allows users to delegate computation while ensuring *secrecy*.



Homomorphic Encryption

Homomorphic encryption allows users to delegate computation while ensuring *secrecy*.

pk

$\xrightarrow[\text{grades}]{\text{encrypted}}$

Untrusted DB

Student	CT
Adam	c_1
Becky	c_2
$\vdots$	$\vdots$
Kevin	c_k

$c_i =$ encryption of ith score

Homomorphic Encryption

Homomorphic encryption allows users to delegate computation while ensuring *secrecy*.

pk

$\xrightarrow[\text{grades}]{\text{encrypted}}$

Untrusted DB

Student	CT
Adam	c_1
Becky	c_2
$\vdots$	$\vdots$
Kevin	c_k

sk

$c_i =$ encryption of ith score

Homomorphic Encryption

Homomorphic encryption allows users to delegate computation while ensuring *secrecy*.

pk

encrypted grades $\longrightarrow$

Untrusted DB

Student	CT
Adam	c_1
Becky	c_2
$\vdots$	$\vdots$
Kevin	c_k

$\longleftarrow$ mean?

sk

c_i = encryption of ith score

Homomorphic encryption allows users to delegate computation while ensuring *secrecy*.

pk

$\xrightarrow[\text{grades}]{\text{encrypted}}$

Untrusted DB

Student	CT
Adam	c_1
Becky	c_2
$\vdots$	$\vdots$
Kevin	c_k

$\xleftarrow{\text{mean?}}$

$\xrightarrow[c]{}$

sk

c_i = encryption of ith score

c = encryption of mean

Homomorphic encryption allows users to delegate computation while ensuring *secrecy*.

pk

encrypted $\longrightarrow$ grades

Untrusted DB

Student	CT
Adam	c_1
Becky	c_2
$\vdots$	$\vdots$
Kevin	c_k

$\xleftarrow{\text{mean?}}$

$\xrightarrow[c]{}$

sk

c_i = encryption of ith score

c = encryption of mean

- Validity: c decrypts to the correct mean.
- Security: no adversary can obtain any info about scores.
- Length efficiency: c is short.
- Privacy: decrypted mean reveals nothing else about data.

Homomorphic Signatures

Homomorphic signatures allow users to delegate computation while ensuring *integrity*.

Homomorphic Signatures

Homomorphic signatures allow users to delegate computation while ensuring *integrity*.

Homomorphic signatures allow users to delegate computation while ensuring *integrity*.

sk

Untrusted DB

Homomorphic Signatures

Homomorphic signatures allow users to delegate computation while ensuring *integrity*.

sk

$\xrightarrow[\text{grades}]{\text{signed}}$

Untrusted DB

Student	Score	Sig
Adam	91	σ_1
Becky	73	σ_2
$\vdots$	$\vdots$	$\vdots$
Kevin	84	σ_k

σ_1 = signature on
(“grades”, 91, “Adam”)

Homomorphic Signatures

Homomorphic signatures allow users to delegate computation while ensuring *integrity*.

sk

signed $\longrightarrow$ grades

Untrusted DB

Student	Score	Sig
Adam	91	σ_1
Becky	73	σ_2
$\vdots$	$\vdots$	$\vdots$
Kevin	84	σ_k

pk

σ_1 = signature on
("grades", 91, "Adam")

Homomorphic Signatures

Homomorphic signatures allow users to delegate computation while ensuring *integrity*.

sk

$\xrightarrow[\text{grades}]{\text{signed}}$

Untrusted DB

Student	Score	Sig
Adam	91	σ_1
Becky	73	σ_2
$\vdots$	$\vdots$	$\vdots$
Kevin	84	σ_k

$\xleftarrow{\text{mean?}}$

pk

$\sigma_1 =$ signature on
(“grades”, 91, “Adam”)

Homomorphic Signatures

Homomorphic signatures allow users to delegate computation while ensuring *integrity*.

sk

$\xrightarrow[\text{grades}]{\text{signed}}$

Untrusted DB

Student	Score	Sig
Adam	91	σ_1
Becky	73	σ_2
$\vdots$	$\vdots$	$\vdots$
Kevin	84	σ_k

$\xleftarrow{\text{mean?}}$

$\xrightarrow[87.3,\ \sigma]{}$

pk

σ_1 = signature on ("grades", 91, "Adam")

σ = signature on ("grades", 87.3, "mean")

Properties of homomorphic signatures

What properties do we want the derived signature σ to have?

$\sigma =$ signature on
(“grades”, 87.3, “mean”)

Properties of homomorphic signatures

What properties do we want the derived signature σ to have?

$\sigma =$ signature on
(“grades”, 87.3, “mean”)

1. Validity: σ authenticates 87.3 as the mean, **and** that the mean was computed correctly.

Properties of homomorphic signatures

What properties do we want the derived signature σ to have?

$$\boxed{\begin{array}{c}\sigma = \text{signature on} \\ (\text{"grades"}, 87.3, \text{"mean"})\end{array}}$$

1. Validity: σ authenticates 87.3 as the mean, **and** that the mean was computed correctly.
2. Unforgeability: no adversary can produce a σ^* that authenticates a different mean.

Properties of homomorphic signatures

What properties do we want the derived signature σ to have?

$\boxed{\sigma = \text{signature on } (\text{“grades”}, 87.3, \text{“mean”})}$

1. Validity: σ authenticates 87.3 as the mean, **and** that the mean was computed correctly.
2. Unforgeability: no adversary can produce a σ^* that authenticates a different mean.
3. Length efficiency: σ is short.

Properties of homomorphic signatures

What properties do we want the derived signature σ to have?

$$\boxed{\begin{array}{c}\sigma = \text{signature on} \\ (\text{“grades”}, 87.3, \text{“mean”})\end{array}}$$

1. Validity: σ authenticates 87.3 as the mean, **and** that the mean was computed correctly.
2. Unforgeability: no adversary can produce a σ^* that authenticates a different mean.
3. Length efficiency: σ is short.
4. Privacy: σ reveals nothing about data other than the mean.

More generally: $\mathcal{F}$-homomorphic signatures

- $\mathcal{F}$ is a set of "admissible" functions on messages.
- τ is a "tag" tying together data from the same set. (like a filename)
    - prevents mixing of data from different sets
- Given *pk*, admissible function $f \in \mathcal{F}$, and signatures on data

$$m_1, \ldots, m_k,$$

anyone can compute a valid signature on

$$(\tau,\ f(m_1, \ldots, m_k),\ \omega(f)),$$

where $\omega(f)$ is an "encoding" or "digest" of the function f.

What are $\mathcal{F}$-homomorphic signatures good for?

$\mathcal{F}$	Application
Linear functions	Mean Linear least-squares fit (fixed x, variable y) Fourier transforms

Applications

What are $\mathcal{F}$-homomorphic signatures good for?

$\mathcal{F}$	Application
Linear functions	Mean Linear least-squares fit (fixed x, variable y) Fourier transforms
Polynomials (bounded degree)	Standard deviation & higher moments Linear least-squares fit (variable x and y)

What are $\mathcal{F}$-homomorphic signatures good for?

$\mathcal{F}$	Application
Linear functions	Mean Linear least-squares fit (fixed x, variable y) Fourier transforms
Polynomials (bounded degree)	Standard deviation & higher moments Linear least-squares fit (variable x and y)
Arbitrary circuits	Non-linear estimators and regression Data mining (decision trees, SVM, etc.)

What are $\mathcal{F}$-homomorphic signatures good for?

$\mathcal{F}$	Application
Linear functions	Mean Linear least-squares fit (fixed x, variable y) Fourier transforms
Polynomials (bounded degree)	Standard deviation & higher moments Linear least-squares fit (variable x and y)
Arbitrary circuits	Non-linear estimators and regression Data mining (decision trees, SVM, etc.)
Subsets	Message redaction

State of the art

How can we compute on encrypted or authenticated data?

$\mathcal{F}$	Hom. encryption	Hom. signatures
Linear functions		
Polynomials (bounded degree)		
Arbitrary circuits		
Subsets		

State of the art

How can we compute on encrypted or authenticated data?

$\mathcal{F}$	Hom. encryption	Hom. signatures
Linear functions	[GM82], [B88], [P99], others	
Polynomials (bounded degree)	[BGN05], [GHV10] (quadratic)	
Arbitrary circuits	[G09], [DGHV10]	
Subsets		

State of the art

How can we compute on encrypted or authenticated data?

$\mathcal{F}$	Hom. encryption	Hom. signatures
Linear functions	[GM82], [B88], [P99], others	[KFM04], [CJL06], [ZKMH07], [BFKW09], [GKKR10], [BF11]
Polynomials (bounded degree)	[BGN05], [GHV10] (quadratic)	
Arbitrary circuits	[G09], [DGHV10]	
Subsets		[JMSW02], others

State of the art

How can we compute on encrypted or authenticated data?

$\mathcal{F}$	Hom. encryption	Hom. signatures
Linear functions	[GM82], [B88], [P99], others	[KFM04], [CJL06], [ZKMH07], [BFKW09], [GKKR10], [BF11]
Polynomials (bounded degree)	[BGN05], [GHV10] (quadratic)	This work
Arbitrary circuits	[G09], [DGHV10]	
Subsets		[JMSW02], others

State of the art

How can we compute on encrypted or authenticated data?

$\mathcal{F}$	Hom. encryption	Hom. signatures
Linear functions	[GM82], [B88], [P99], others	[KFM04], [CJL06], [ZKMH07], [BFKW09], [GKKR10], [BF11]
Polynomials (bounded degree)	[BGN05], [GHV10] (quadratic)	This work
Arbitrary circuits	[G09], [DGHV10]	
Subsets		[JMSW02], others

Specifically, we construct secure, length-efficient, $\mathcal{F}$-homomorphic signatures for

$$\mathcal{F} = \{\text{polynomials of bounded degree with small coefficients}\}$$

Application: Least Squares Fits

Least squares fits — the basics

For a data set $\{(x_i, y_i)\}_{i=1}^{k}$, the degree d least squares fit is a polynomial

$$f(x) = c_0 + c_1 x + \cdots + c_d x^d$$

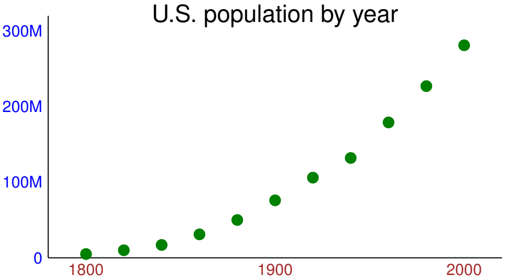
that "best" approximates the y values.

Least squares fits — the basics

For a data set $\{(x_i, y_i)\}_{i=1}^{k}$, the degree d least squares fit is a polynomial

$$f(x) = c_0 + c_1 x + \cdots + c_d x^d$$

that "best" approximates the y values.

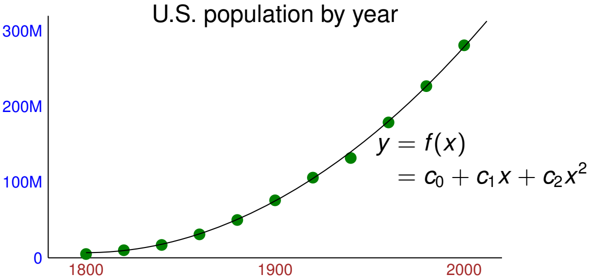


Least squares fits — the basics

For a data set $\{(x_i, y_i)\}_{i=1}^{k}$, the degree d least squares fit is a polynomial

$$f(x) = c_0 + c_1 x + \cdots + c_d x^d$$

that "best" approximates the y values.

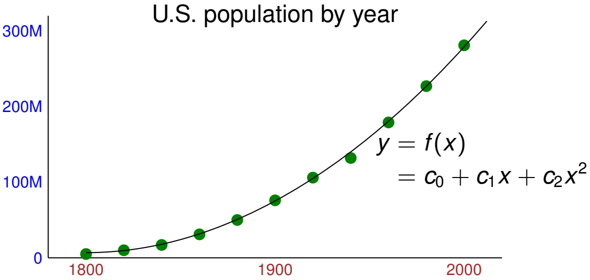


Least squares fits — the basics

For a data set $\{(x_i, y_i)\}_{i=1}^k$, the degree d least squares fit is a polynomial

$$f(x) = c_0 + c_1 x + \cdots + c_d x^d$$

that "best" approximates the y values.

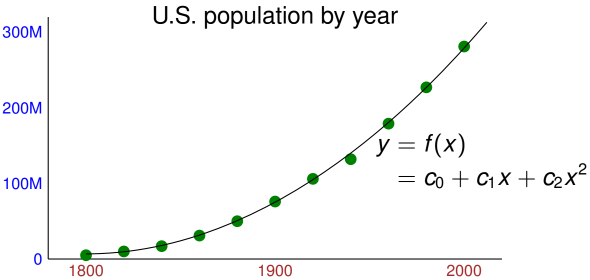


Formula:

$$\vec{c} = (X^t X)^{-1} X^t \vec{y}$$

$\vec{c}$ = vector of coefficients of $f(x)$,
X = Vandermonde matrix of x values,
$\vec{y}$ = vector of y values.

Authenticating a least-squares fit (x-values only)

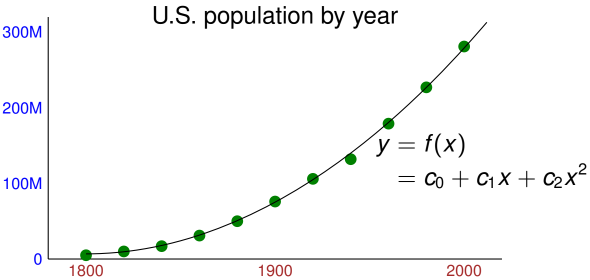


Formula:

$$\vec{c} = (X^t X)^{-1} X^t \vec{y}$$

$\vec{c}$ = vector of coefficients of $f(x)$,
X = Vandermonde matrix of x values,
$\vec{y}$ = vector of y values.

Authenticating a least-squares fit (x-values only)



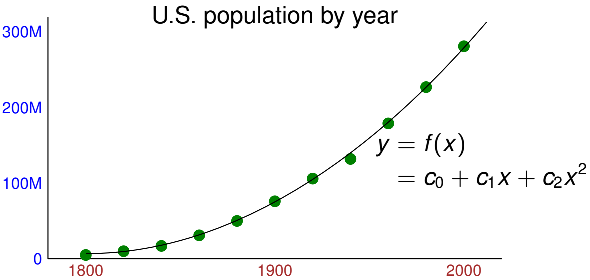
Formula:

$$\vec{c} = (X^t X)^{-1} X^t \vec{y}$$

$\vec{c}$ = vector of coefficients of $f(x)$,
X = Vandermonde matrix of x values,
$\vec{y}$ = vector of y values.

If x values are fixed, then $\vec{c}$ is linear function of y values.

Authenticating a least-squares fit (x-values only)



Formula:

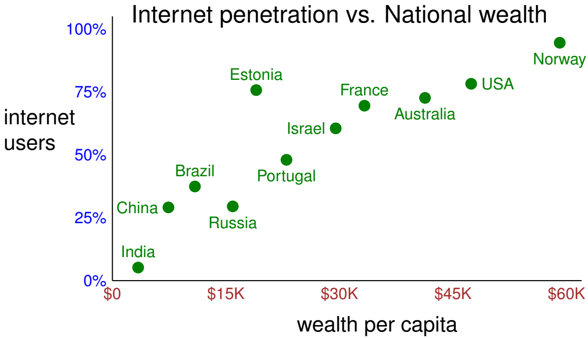
$$\vec{c} = (X^t X)^{-1} X^t \vec{y}$$

$\vec{c}$ = vector of coefficients of $f(x)$,
X = Vandermonde matrix of x values,
$\vec{y}$ = vector of y values.

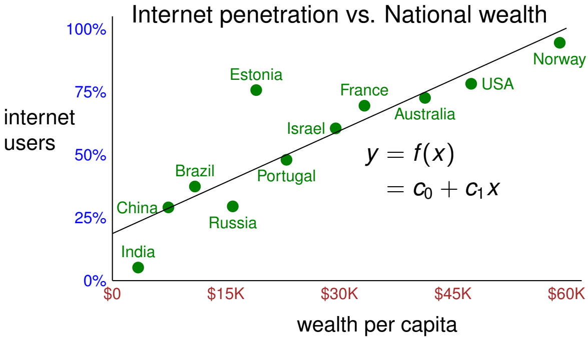
If x values are fixed, then $\vec{c}$ is linear function of y values.

- Census bureau stores signed population counts on server using linearly homomorphic signature.
- Server can authenticate coefficients of least-squares fit.

Authenticating a least-squares fit (x and y values)



Authenticating a least-squares fit (x and y values)

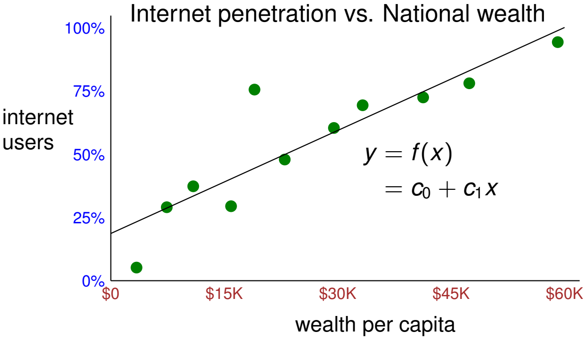


Formula:

$$\vec{c} = (X^t X)^{-1} X^t \vec{y}$$

$\vec{c}$ = vector of coefficients of $f(x)$,
X = Vandermonde matrix of x values,
$\vec{y}$ = vector of y values.

Authenticating a least-squares fit (x and y values)



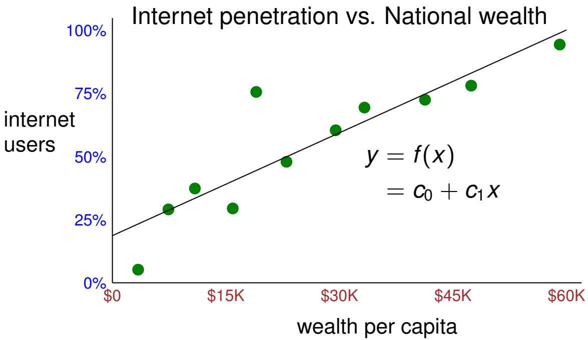
Formula:

$$\vec{c} = (X^t X)^{-1} X^t \vec{y}$$

$\vec{c}$ = vector of coefficients of $f(x)$,
X = Vandermonde matrix of x values,
$\vec{y}$ = vector of y values.

- Coefficients c_j are rational functions of sampled x and y values.

Authenticating a least-squares fit (x and y values)

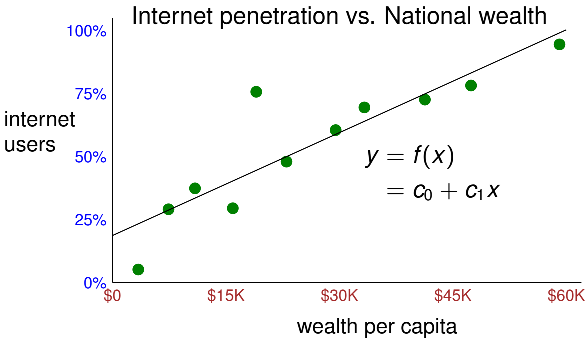


Formula:

$$\vec{c} = (X^t X)^{-1} X^t \vec{y}$$

$\vec{c}$ = vector of coefficients of $f(x)$,
X = Vandermonde matrix of x values,
$\vec{y}$ = vector of y values.

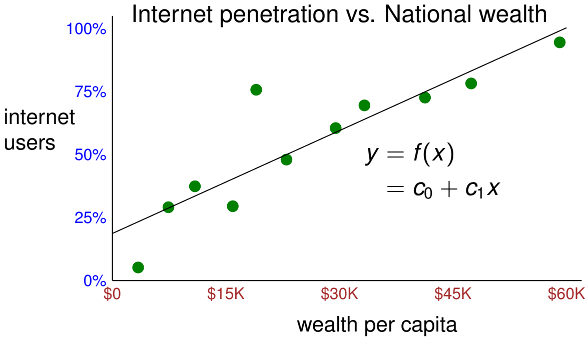
- Coefficients c_j are rational functions of sampled x and y values.
- However: $\det(X^t X) \cdot c_j$ are polynomial functions of x and y.



Formula: $\det(X^t X) \cdot c_j =$ polynomial in $\{x_i, y_i\}$

- Coefficients c_j are rational functions of sampled x and y values.
- However: $\det(X^t X) \cdot c_j$ are polynomial functions of x and y.

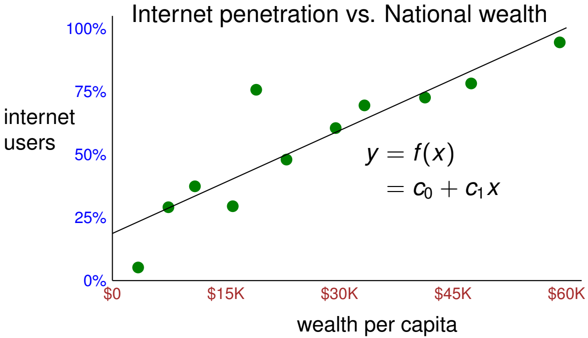
Authenticating a least-squares fit (x and y values)



Formula: $\det(X^t X) \cdot c_j =$ polynomial in $\{x_i, y_i\}$

- United Nations stores signed data on server using polynomially homomorphic signature.

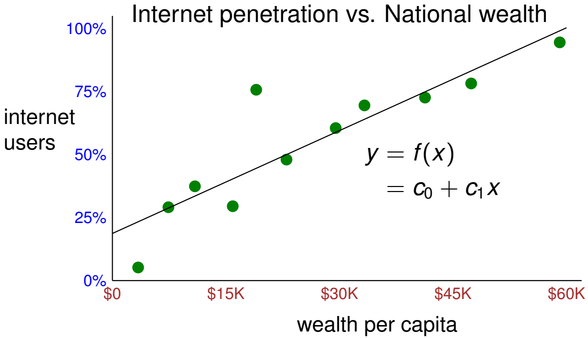
Authenticating a least-squares fit (x and y values)



Formula: $\det(X^t X) \cdot c_j =$ polynomial in $\{x_i, y_i\}$

- United Nations stores signed data on server using polynomially homomorphic signature.
- Server can authenticate $\det(X^t X)$ and $\det(X^t X) \cdot \vec{c}$.

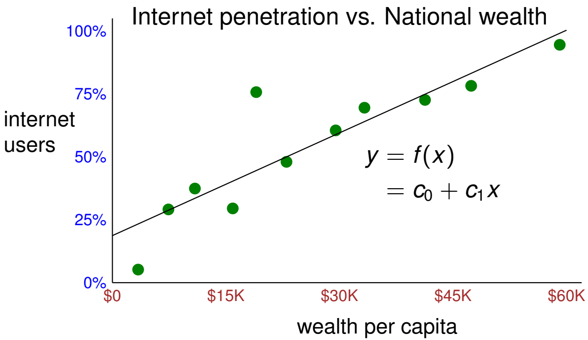
Authenticating a least-squares fit (x and y values)



Formula: $\det(X^t X) \cdot c_j =$ polynomial in $\{x_i, y_i\}$

- United Nations stores signed data on server using polynomially homomorphic signature.
- Server can authenticate $\det(X^t X)$ and $\det(X^t X) \cdot \vec{c}$.
- User can compute least-squares fit from server's values.

Authenticating a least-squares fit (x and y values)



Formula: $\det(X^t X) \cdot c_j =$ polynomial in $\{x_i, y_i\}$

- United Nations stores signed data on server using polynomially homomorphic signature.
- Server can authenticate $\det(X^t X)$ and $\det(X^t X) \cdot \vec{c}$.
- User can compute least-squares fit from server's values.
- Linear fit can be computed using degree 3 polynomials.

Linearly Homomorphic Signatures

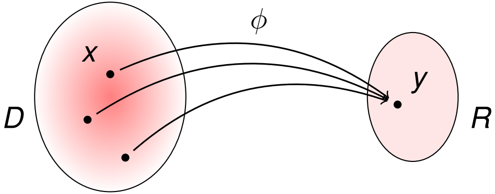
Building block: GPV Signatures

Key idea: preimage sampleable trapdoor function

Building block: GPV Signatures

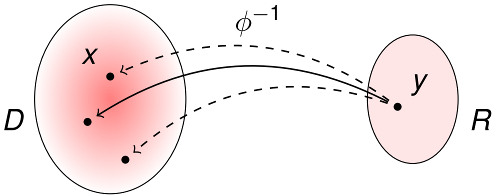
Key idea: preimage sampleable trapdoor function

- Public function $\phi\colon D \to R$ with secret "trapdoor" ϕ^{-1}



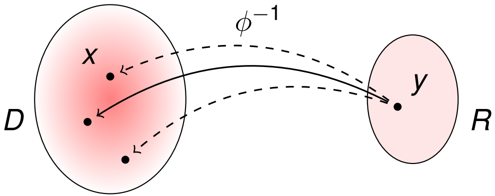
Key idea: preimage sampleable trapdoor function

- Public function $\phi \colon D \to R$ with secret "trapdoor" ϕ^{-1}



Key idea: preimage sampleable trapdoor function

- Public function $\phi\colon D \to R$ with secret "trapdoor" ϕ^{-1}

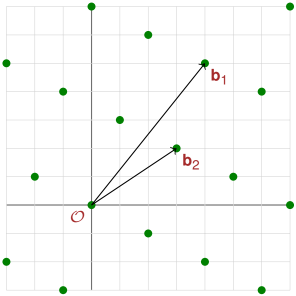


- "Hash and sign:" $pk = \phi$, $sk = \phi^{-1}$, hash $H\colon \{0,1\}^* \to R$

$$\begin{aligned}\mathrm{Sign}(m) &:= \phi^{-1}(H(m))\\ \mathrm{Verify}(\sigma) &: \phi(\sigma) \stackrel{?}{=} H(m)\end{aligned}$$

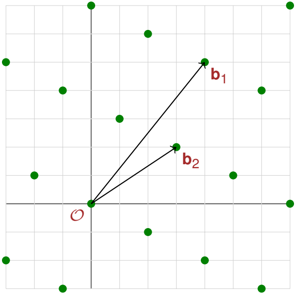
GPV Signatures, concretely

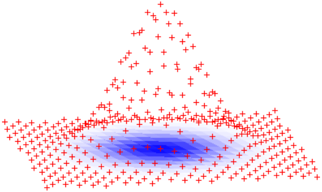
- $\Lambda \subset \mathbb{Z}^n$ a lattice (full-rank additive subgroup), defined by basis.



GPV Signatures, concretely

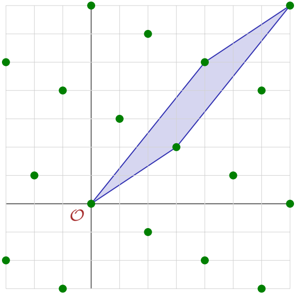
- $\Lambda \subset \mathbb{Z}^n$ a lattice (full-rank additive subgroup), defined by basis.
- $D =$ short vectors in $\mathbb{Z}^n$, with Gaussian distribution.

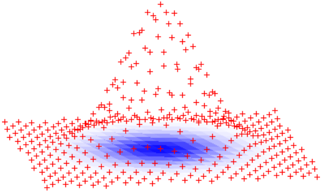




GPV Signatures, concretely

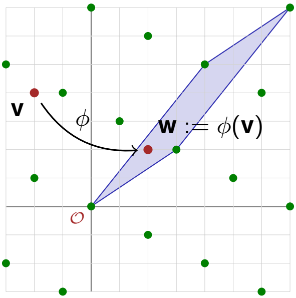
- $\Lambda \subset \mathbb{Z}^n$ a lattice (full-rank additive subgroup), defined by basis.
- $D =$ short vectors in $\mathbb{Z}^n$, with Gaussian distribution.
- $R = \mathbb{Z}^n/\Lambda$ (fix unique representatives)

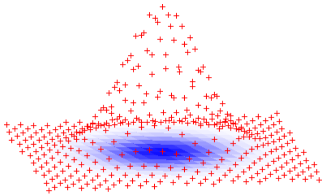




GPV Signatures, concretely

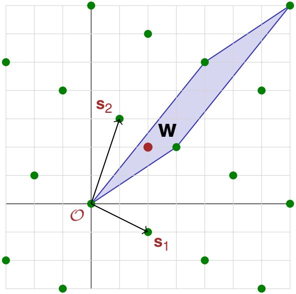
- $\Lambda \subset \mathbb{Z}^n$ a lattice (full-rank additive subgroup), defined by basis.
- $D =$ short vectors in $\mathbb{Z}^n$, with Gaussian distribution.
- $R = \mathbb{Z}^n/\Lambda$ (fix unique representatives)
- Trapdoor function ϕ: $\mathbf{v} \mapsto (\mathbf{v} \bmod \Lambda)$ i.e., move $\mathbf{v}$ into a fundamental parallelepiped.

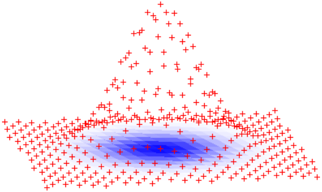




GPV Signatures, concretely

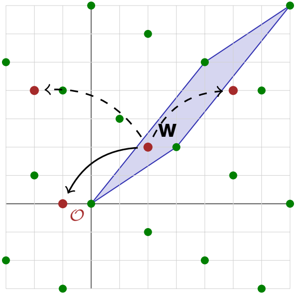
- $\Lambda \subset \mathbb{Z}^n$ a lattice (full-rank additive subgroup), defined by basis.
- $D =$ short vectors in $\mathbb{Z}^n$, with Gaussian distribution.
- $R = \mathbb{Z}^n/\Lambda$ (fix unique representatives)
- Trapdoor function $\phi \colon \mathbf{v} \mapsto (\mathbf{v} \bmod \Lambda)$ i.e., move $\mathbf{v}$ into a fundamental parallelepiped.
- GPV: algorithm to sample short vectors in $\phi^{-1}(\mathbf{w}) = \Lambda + \mathbf{w}$ given a “short” basis of Λ.

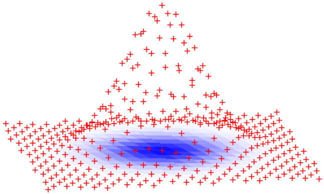




GPV Signatures, concretely

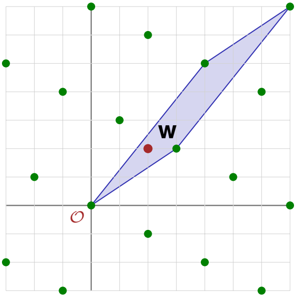
- $\Lambda \subset \mathbb{Z}^n$ a lattice (full-rank additive subgroup), defined by basis.
- D = short vectors in $\mathbb{Z}^n$, with Gaussian distribution.
- $R = \mathbb{Z}^n/\Lambda$ (fix unique representatives)
- Trapdoor function ϕ: $\mathbf{v} \mapsto (\mathbf{v} \bmod \Lambda)$ i.e., move $\mathbf{v}$ into a fundamental parallelepiped.
- GPV: algorithm to sample short vectors in $\phi^{-1}(\mathbf{w}) = \Lambda + \mathbf{w}$ given a "short" basis of Λ.

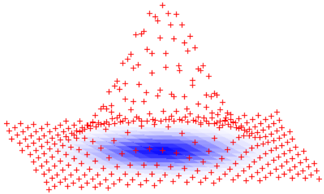




GPV Signatures, concretely

- $\Lambda \subset \mathbb{Z}^n$ a lattice (full-rank additive subgroup), defined by basis.
- $D =$ short vectors in $\mathbb{Z}^n$, with Gaussian distribution.
- $R = \mathbb{Z}^n / \Lambda$ (fix unique representatives)
- Trapdoor function $\phi \colon \mathbf{v} \mapsto (\mathbf{v} \bmod \Lambda)$ i.e., move $\mathbf{v}$ into a fundamental parallelepiped.
- GPV: algorithm to sample short vectors in $\phi^{-1}(\mathbf{w}) = \Lambda + \mathbf{w}$ given a "short" basis of Λ.
- Sampling from $\Lambda + \mathbf{w}$ without short basis is hard. (How hard depends on Gaussian parameter.)





Linearly Homomorphic Signatures: Key Idea #1

Idea: instead of hashing the messages to $R = \mathbb{Z}^n/\Lambda$,
let the message space be *R* itself.

Linearly Homomorphic Signatures: Key Idea #1

Idea: instead of hashing the messages to $R = \mathbb{Z}^n/\Lambda$,
let the message space be R itself.

GPV sign/verify algorithms: $\quad H\colon \{0,1\}^* \to \mathbb{Z}^n/\Lambda$

$$\begin{aligned} \text{Sign}(m) &:= \text{short vector in } (\Lambda + H(m)) \\ \text{Verify}(\sigma) &:= 1 \quad \text{iff} \quad \sigma \text{ is short}, \;\; \sigma \bmod \Lambda = H(m) \end{aligned}$$

Linearly Homomorphic Signatures: Key Idea #1

Idea: instead of hashing the messages to $R = \mathbb{Z}^n/\Lambda$,
let the message space be R itself.

New sign/verify algorithms: $m \in \mathbb{Z}^n/\Lambda$

$$\begin{aligned} \text{Sign}(m) &:= \text{short vector in } (\Lambda + m) \\ \text{Verify}(\sigma) &:= 1 \quad \text{iff} \quad \sigma \text{ is short}, \;\; \sigma \bmod \Lambda = m \end{aligned}$$

Linearly Homomorphic Signatures: Key Idea #1

Idea: instead of hashing the messages to $R = \mathbb{Z}^n/\Lambda$, let the message space be R itself.

New sign/verify algorithms: $m \in \mathbb{Z}^n/\Lambda$

$$\begin{aligned} \text{Sign}(m) &:= \text{short vector in } (\Lambda + m) \\ \text{Verify}(\sigma) &:= 1 \quad \text{iff} \quad \sigma \text{ is short}, \ \ \sigma \bmod \Lambda = m \end{aligned}$$

Homomorphic property: ϕ is a linear map, so adding signatures corresponds to adding messages.

Linearly Homomorphic Signatures: Key Idea #1

Idea: instead of hashing the messages to $R = \mathbb{Z}^n/\Lambda$,
let the message space be R itself.

New sign/verify algorithms: $m \in \mathbb{Z}^n/\Lambda$

$$\begin{aligned} \text{Sign}(m) &:= \text{short vector in } (\Lambda + m) \\ \text{Verify}(\sigma) &:= 1 \quad \text{iff} \quad \sigma \text{ is short}, \;\; \sigma \bmod \Lambda = m \end{aligned}$$

Homomorphic property: ϕ is a linear map, so adding signatures corresponds to adding messages.

- Suppose σ_1, σ_2 are signatures on m_1, m_2
  $\Rightarrow \sigma_i$ short, $\sigma_i \bmod \Lambda = m_i$.

Linearly Homomorphic Signatures: Key Idea #1

Idea: instead of hashing the messages to $R = \mathbb{Z}^n/\Lambda$,
let the message space be R itself.

New sign/verify algorithms: $m \in \mathbb{Z}^n/\Lambda$

$$\begin{aligned} \text{Sign}(m) &:= \text{short vector in } (\Lambda + m) \\ \text{Verify}(\sigma) &:= 1 \quad \text{iff} \quad \sigma \text{ is short}, \ \ \sigma \bmod \Lambda = m \end{aligned}$$

Homomorphic property: ϕ is a linear map, so adding signatures corresponds to adding messages.

- Suppose σ_1, σ_2 are signatures on m_1, m_2
  $\Rightarrow \sigma_i$ short, $\sigma_i \bmod \Lambda = m_i$.
- For $a, b \in \mathbb{Z}$, define signature on $am_1 + bm_2$ to be
  $$\sigma := a\sigma_1 + b\sigma_2.$$
  $\Rightarrow \sigma$ is short (if a, b small), $\sigma \bmod \Lambda = am_1 + bm_2$.

Problem: Removing hash function destroys security

Valid signature doesn't imply function was computed correctly.

Valid signature doesn't imply function was computed correctly.

sk

signed
grades →

Untrusted DB

Student	Score	Sig
1	91	σ_1
2	73	σ_2
⋮	⋮	⋮
k	84	σ_k

← mean?

pk

Valid signature doesn't imply function was computed correctly.

sk

$\xrightarrow[\text{grades}]{\text{signed}}$

Untrusted DB

Student	Score	Sig
1	91	σ_1
2	73	σ_2
$\vdots$	$\vdots$	$\vdots$
k	84	σ_k

$\xleftarrow{\text{mean?}}$

$\xrightarrow[87.3,\ \sigma]{}$

pk

- Honest DB outputs $87.3 = \frac{1}{k}\sum s_i$ and signature $\sigma = \frac{1}{k}\sum \sigma_i$.

Problem: Removing hash function destroys security

Valid signature doesn't imply function was computed correctly.

sk

$\xrightarrow[\text{grades}]{\text{signed}}$

Untrusted DB

Student	Score	Sig
1	91	σ_1
2	73	σ_2
⋮	⋮	⋮
k	84	σ_k

$\xleftarrow{\text{mean?}}$

$\longrightarrow$

18.0, σ

pk

- Honest DB outputs $87.3 = \frac{1}{k}\sum s_i$ and signature $\sigma = \frac{1}{k}\sum \sigma_i$.
- Malicious DB outputs $18.0 = s_1 - s_2$ and signature $\sigma = \sigma_1 - \sigma_2$.

Problem: Removing hash function destroys security

Valid signature doesn't imply function was computed correctly.

sk

$\xrightarrow{\text{signed}}$ grades

Untrusted DB

Student	Score	Sig
1	91	σ_1
2	73	σ_2
⋮	⋮	⋮
k	84	σ_k

$\xleftarrow{\text{mean?}}$

$\longrightarrow$ 18.0, σ

pk

- Honest DB outputs $87.3 = \frac{1}{k}\sum s_i$ and signature $\sigma = \frac{1}{k}\sum \sigma_i$.
- Malicious DB outputs $18.0 = s_1 - s_2$ and signature $\sigma = \sigma_1 - \sigma_2$.
- σ authenticates 18, but 18 is not the mean!

Linearly Homomorphic Signatures: Key Idea #2

Use a second lattice to authenticate functions:

- $\Lambda_2 \subset \mathbb{Z}^n$ distinct from $\Lambda_1 := \Lambda$.
  - require $\Lambda_1 + \Lambda_2 = \mathbb{Z}^n$
- Map $\phi_2 \colon \mathbb{Z}^n \to \mathbb{Z}^n / \Lambda_2$ given by $\phi_2(\mathbf{v}) := \mathbf{v} \bmod \Lambda_2$.

Linearly Homomorphic Signatures: Key Idea #2

Use a second lattice to authenticate functions:

- $\Lambda_2 \subset \mathbb{Z}^n$ distinct from $\Lambda_1 := \Lambda$.
  - require $\Lambda_1 + \Lambda_2 = \mathbb{Z}^n$
- Map $\phi_2 \colon \mathbb{Z}^n \to \mathbb{Z}^n/\Lambda_2$ given by $\phi_2(\mathbf{v}) := \mathbf{v} \bmod \Lambda_2$.

"Encode" functions f as elements $\omega(f) \in \mathbb{Z}^n/\Lambda_2$.
Sign functions by computing

$$\text{Sign}(f) := \text{short vector in } (\Lambda_2 + \omega(f)).$$

Linearly Homomorphic Signatures: Key Idea #2

Use a second lattice to authenticate functions:

- $\Lambda_2 \subset \mathbb{Z}^n$ distinct from $\Lambda_1 := \Lambda$.
  - require $\Lambda_1 + \Lambda_2 = \mathbb{Z}^n$
- Map $\phi_2 \colon \mathbb{Z}^n \to \mathbb{Z}^n/\Lambda_2$ given by $\phi_2(\mathbf{v}) := \mathbf{v} \bmod \Lambda_2$.

"Encode" functions f as elements $\omega(f) \in \mathbb{Z}^n/\Lambda_2$.
Sign functions by computing

$$\mathrm{Sign}(f) := \text{short vector in } (\Lambda_2 + \omega(f)).$$

If "encoding" $\omega(\cdot)$ is linear, (i.e., $\omega(f) + \omega(g) = \omega(f+g)$)
then signature is a linear operator on the space of functions.

How do we encode functions?

Ingredients:

- k := number of messages input to a function.
- τ := "tag" that ties together messages in same data set.
- Hash function $H\colon \{0,1\}^* \to (\mathbb{Z}^n/\Lambda_2)^k$ maps $\tau \mapsto (\alpha_1, \ldots, \alpha_k)$.

How do we encode functions?

Ingredients:

- $k :=$ number of messages input to a function.
- $\tau :=$ "tag" that ties together messages in same data set.
- Hash function $H\colon \{0,1\}^* \to (\mathbb{Z}^n/\Lambda_2)^k$ maps $\tau \mapsto (\alpha_1, \ldots, \alpha_k)$.
- Observation:

$$\left\{ \begin{array}{c} \text{Linear functions} \\ \text{in } k \text{ variables} \end{array} \right\} \quad \text{generated by} \quad \left\{ \begin{array}{c} \text{"projections" } \pi_i : \\ \pi_i(m_1, \ldots, m_k) = m_i \end{array} \right\}_{i=1}^{k}$$

How do we encode functions?

Ingredients:

- $k :=$ number of messages input to a function.
- $\tau :=$ "tag" that ties together messages in same data set.
- Hash function $H\colon \{0,1\}^* \to (\mathbb{Z}^n/\Lambda_2)^k$ maps $\tau \mapsto (\alpha_1, \ldots, \alpha_k)$.
- Observation:

$$\left\{ \begin{array}{c} \text{Linear functions} \\ \text{in } k \text{ variables} \end{array} \right\} \quad \text{generated by} \quad \left\{ \begin{array}{c} \text{"projections" } \pi_i : \\ \pi_i(m_1, \ldots, m_k) = m_i \end{array} \right\}_{i=1}^{k}$$

Define "encoding" $\omega\colon \mathcal{F} \to \mathbb{Z}^n/\Lambda_2$ by

$$f = \sum c_i \pi_i \quad \mapsto \quad \omega(f) = \sum c_i \alpha_i = f(\alpha_1, \ldots, \alpha_k).$$

How do we encode functions?

Ingredients:

- $k :=$ number of messages input to a function.
- $\tau :=$ "tag" that ties together messages in same data set.
- Hash function $H\colon \{0,1\}^* \to (\mathbb{Z}^n/\Lambda_2)^k$ maps $\tau \mapsto (\alpha_1, \ldots, \alpha_k)$.
- Observation:

$$\left\{ \begin{array}{c} \text{Linear functions} \\ \text{in } k \text{ variables} \end{array} \right\} \quad \text{generated by} \quad \left\{ \begin{array}{c} \text{“projections” } \pi_i: \\ \pi_i(m_1, \ldots, m_k) = m_i \end{array} \right\}_{i=1}^{k}$$

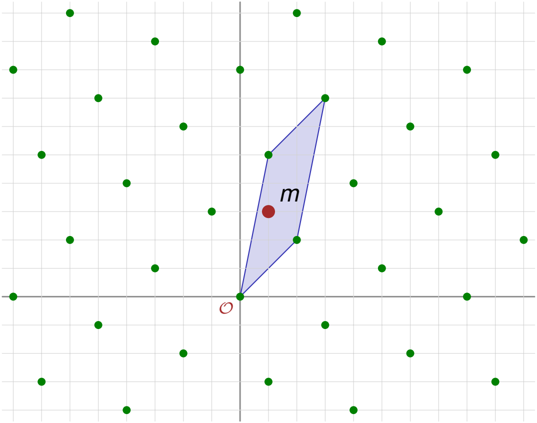
Define "encoding" $\omega\colon \mathcal{F} \to \mathbb{Z}^n/\Lambda_2$ by

$$f = \sum c_i \pi_i \quad \mapsto \quad \omega(f) = \sum c_i \alpha_i = f(\alpha_1, \ldots, \alpha_k).$$

- c_i are small integers.
- "encoding" $\omega(f)$ much shorter than description of f.

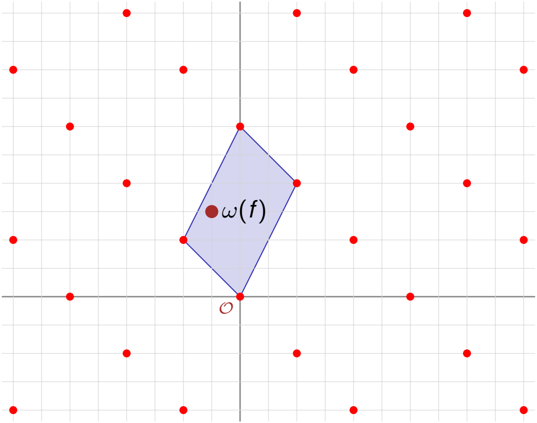
“Intersection method” binds messages to functions

- Messages $m \in \mathbb{Z}^n/\Lambda_1$.



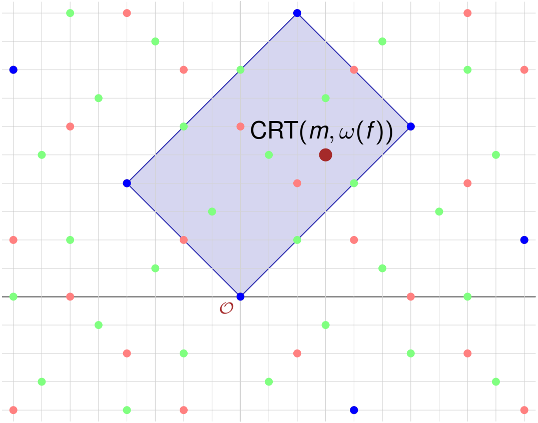
"Intersection method" binds messages to functions

- Messages $m \in \mathbb{Z}^n/\Lambda_1$.
- Functions $f = \sum c_i \pi_i$ encoded as $\omega(f) = \sum c_i \alpha_i \in \mathbb{Z}^n/\Lambda_2$.
  (α_i defined by tag τ.)



"Intersection method" binds messages to functions

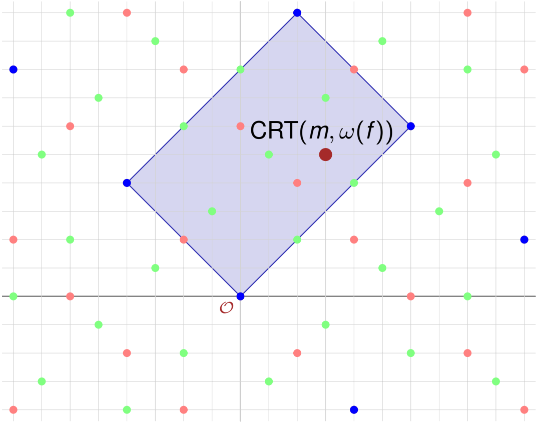
- Messages $m \in \mathbb{Z}^n/\Lambda_1$.
- Functions $f = \sum c_i \pi_i$ encoded as $\omega(f) = \sum c_i \alpha_i \in \mathbb{Z}^n/\Lambda_2$.
  (α_i defined by tag τ.)
- Pair $(m, \omega(f))$ gives unique element of $\mathbb{Z}^n/\Lambda_1 \cap \Lambda_2$.



$$\mathrm{CRT}(m, \omega(f))$$
$$= m \bmod \Lambda_1$$
$$= \omega(f) \bmod \Lambda_2$$

Signing a message-function pair

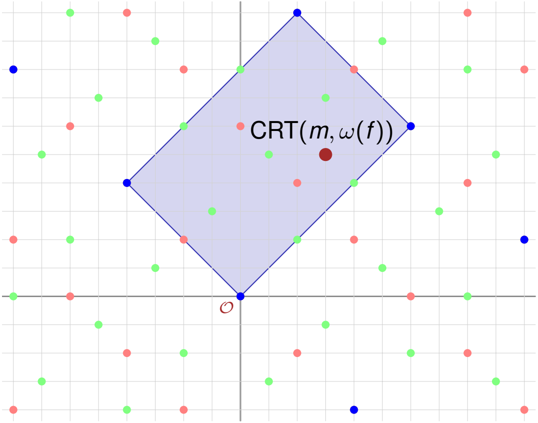
- Pair $(m, \omega(f))$ gives unique element of $\mathbb{Z}^n / \Lambda_1 \cap \Lambda_2$.



$$\begin{aligned} &\mathrm{CRT}(m, \omega(f)) \\ &= m \bmod \Lambda_1 \\ &= \omega(f) \bmod \Lambda_2 \end{aligned}$$

Signing a message-function pair

- Pair $(m, \omega(f))$ gives unique element of $\mathbb{Z}^n/\Lambda_1 \cap \Lambda_2$.



$$\mathrm{CRT}(m, \omega(f)) = m \bmod \Lambda_1 = \omega(f) \bmod \Lambda_2$$

$$\begin{aligned} \mathrm{Sign}(m) &:= \text{short vector in } (\Lambda_1 \cap \Lambda_2) + \mathrm{CRT}(m, \omega(f)) \\ \mathrm{Verify}(\sigma) &:= 1 \quad \text{iff} \quad (\sigma \bmod \Lambda_1 = m) \text{ and } (\sigma \bmod \Lambda_2 = \omega(f)) \\ &\qquad \text{and } \sigma \text{ is short} \end{aligned}$$

Linearly homomorphic signature scheme

- KeyGen(n):
    - pk = Lattices $\Lambda_1, \Lambda_2 \subset \mathbb{Z}^n$, Gaussian parameter β
    - sk = short basis of $\Lambda_1 \cap \Lambda_2$
    - $H\colon \{0,1\}^* \to (\mathbb{Z}^n/\Lambda_2)^k, \quad H(\tau) = (\alpha_1, \ldots, \alpha_k)$.

Linearly homomorphic signature scheme

- KeyGen(n):
  - pk = Lattices $\Lambda_1, \Lambda_2 \subset \mathbb{Z}^n$, Gaussian parameter β
  - sk = short basis of $\Lambda_1 \cap \Lambda_2$
  - $H \colon \{0,1\}^* \to (\mathbb{Z}^n/\Lambda_2)^k, \quad H(\tau) = (\alpha_1, \ldots, \alpha_k)$.
- Sign(τ, m_i, π_i): compute short vector σ_i in $\Lambda_1 \cap \Lambda_2 + \mathrm{CRT}(m_i, \alpha_i)$.
  - π_i = ith projection function

Linearly homomorphic signature scheme

- KeyGen(n):
  - pk = Lattices $\Lambda_1, \Lambda_2 \subset \mathbb{Z}^n$, Gaussian parameter β
  - sk = short basis of $\Lambda_1 \cap \Lambda_2$
  - $H\colon \{0,1\}^* \to (\mathbb{Z}^n/\Lambda_2)^k, \quad H(\tau) = (\alpha_1, \ldots, \alpha_k)$.
- Sign(τ, m_i, π_i): compute short vector σ_i in $\Lambda_1 \cap \Lambda_2 + \mathrm{CRT}(m_i, \alpha_i)$.
  - π_i = ith projection function
- Evaluate($f = \sum c_i \pi_i, (\sigma_1, \ldots, \sigma_k)$): compute $\sigma = \sum c_i \sigma_i$.

Linearly homomorphic signature scheme

- KeyGen(n):
  - pk = Lattices $\Lambda_1, \Lambda_2 \subset \mathbb{Z}^n$, Gaussian parameter β
  - sk = short basis of $\Lambda_1 \cap \Lambda_2$
  - $H\colon \{0,1\}^* \to (\mathbb{Z}^n/\Lambda_2)^k, \quad H(\tau) = (\alpha_1, \ldots, \alpha_k)$.
- Sign(τ, m_i, π_i): compute short vector σ_i in $\Lambda_1 \cap \Lambda_2 + \mathrm{CRT}(m_i, \alpha_i)$.
  - π_i = ith projection function
- Evaluate($f = \sum c_i\pi_i, (\sigma_1, \ldots, \sigma_k)$): compute $\sigma = \sum c_i\sigma_i$.
- Verify($\tau, \sigma, m, f = \sum c_i\pi_i$): Accept if
  1. $\sigma \bmod \Lambda_1 = m$,
  2. $\sigma \bmod \Lambda_2 = \omega(f) = \sum c_i\alpha_i$,
  3. σ sufficiently short.

Concrete example

- Lattices:

Λ_1	Λ_2	$\Lambda_1 \cap \Lambda_2$

Concrete example

- Lattices:

$\Lambda_1 = p\mathbb{Z}^n$ p small prime	Λ_2	$\Lambda_1 \cap \Lambda_2$

Concrete example

- Lattices:

$\Lambda_1 = p\mathbb{Z}^n$ p small prime	$\Lambda_2 = \Lambda_q^\perp(\mathbf{A}) =$ $\{\mathbf{x} \in \mathbb{Z}^n : \mathbf{A} \cdot \mathbf{x} = 0 \bmod q\}$ $q \neq p$ prime, $\mathbf{A} \in \mathbb{F}_q^{n' \times n}$	$\Lambda_1 \cap \Lambda_2$

  - Can sample random $\Lambda_q^\perp(\mathbf{A})$ with short basis $\mathbf{B}$ [A99,AP09].

Concrete example

- Lattices:

$\Lambda_1 = p\mathbb{Z}^n$	$\Lambda_2 = \Lambda_q^\perp(\mathbf{A}) =$	$\Lambda_1 \cap \Lambda_2 = p \cdot \Lambda_q^\perp(\mathbf{A})$
p small prime	$\{\mathbf{x} \in \mathbb{Z}^n : \mathbf{A} \cdot \mathbf{x} = 0 \bmod q\}$	short basis is $p \cdot \mathbf{B}$
	$q \neq p$ prime, $\mathbf{A} \in \mathbb{F}_q^{n' \times n}$	

  - Can sample random $\Lambda_q^\perp(\mathbf{A})$ with short basis $\mathbf{B}$ [A99,AP09].

Concrete example

- Lattices:

$\Lambda_1 = p\mathbb{Z}^n$ p small prime	$\Lambda_2 = \Lambda_q^\perp(\mathbf{A}) =$ $\{\mathbf{x} \in \mathbb{Z}^n : \mathbf{A} \cdot \mathbf{x} = 0 \bmod q\}$ $q \neq p$ prime, $\mathbf{A} \in \mathbb{F}_q^{n' \times n}$	$\Lambda_1 \cap \Lambda_2 = p \cdot \Lambda_q^\perp(\mathbf{A})$ short basis is $p \cdot \mathbf{B}$

  - Can sample random $\Lambda_q^\perp(\mathbf{A})$ with short basis $\mathbf{B}$ [A99,AP09].

- Message space: $\mathbb{Z}^n/p\mathbb{Z}^n = \mathbb{F}_p^n$.

Concrete example

- Lattices:

$\Lambda_1 = p\mathbb{Z}^n$ p small prime	$\Lambda_2 = \Lambda_q^\perp(\mathbf{A}) =$ $\{\mathbf{x} \in \mathbb{Z}^n : \mathbf{A} \cdot \mathbf{x} = 0 \bmod q\}$ $q \neq p$ prime, $\mathbf{A} \in \mathbb{F}_q^{n' \times n}$	$\Lambda_1 \cap \Lambda_2 = p \cdot \Lambda_q^\perp(\mathbf{A})$ short basis is $p \cdot \mathbf{B}$

  - Can sample random $\Lambda_q^\perp(\mathbf{A})$ with short basis $\mathbf{B}$ [A99,AP09].

- Message space: $\mathbb{Z}^n / p\mathbb{Z}^n = \mathbb{F}_p^n$.
- Admissible functions $f = \sum c_i \pi_i$, $c_i \in \mathbb{F}_p$:
  $\mathbb{F}_p$-linear combinations of k vectors in $\mathbb{F}_p^n$.

Concrete example

- Lattices:

$\Lambda_1 = p\mathbb{Z}^n$ p small prime	$\Lambda_2 = \Lambda_q^\perp(\mathbf{A}) =$ $\{\mathbf{x} \in \mathbb{Z}^n : \mathbf{A} \cdot \mathbf{x} = 0 \bmod q\}$ $q \neq p$ prime, $\mathbf{A} \in \mathbb{F}_q^{n' \times n}$	$\Lambda_1 \cap \Lambda_2 = p \cdot \Lambda_q^\perp(\mathbf{A})$ short basis is $p \cdot \mathbf{B}$

  - Can sample random $\Lambda_q^\perp(\mathbf{A})$ with short basis $\mathbf{B}$ [A99,AP09].

- Message space: $\mathbb{Z}^n / p\mathbb{Z}^n = \mathbb{F}_p^n$.
- Admissible functions $f = \sum c_i \pi_i$, $c_i \in \mathbb{F}_p$:
  $\mathbb{F}_p$-linear combinations of k vectors in $\mathbb{F}_p^n$.

Signature scheme signs k vectors $\mathbf{v}_i \in \mathbb{F}_p^n$ and can authenticate any $\mathbb{F}_p$-linear combination of the $\mathbf{v}_i$.

Concrete example

- Lattices:

$\Lambda_1 = p\mathbb{Z}^n$	$\Lambda_2 = \Lambda_q^\perp(\mathbf{A}) =$	$\Lambda_1 \cap \Lambda_2 = p \cdot \Lambda_q^\perp(\mathbf{A})$
p small prime	$\{\mathbf{x} \in \mathbb{Z}^n : \mathbf{A} \cdot \mathbf{x} = 0 \bmod q\}$	short basis is $p \cdot \mathbf{B}$
	$q \neq p$ prime, $\mathbf{A} \in \mathbb{F}_q^{n' \times n}$	

  - Can sample random $\Lambda_q^\perp(\mathbf{A})$ with short basis $\mathbf{B}$ [A99,AP09].

- Message space: $\mathbb{Z}^n / p\mathbb{Z}^n = \mathbb{F}_p^n$.
- Admissible functions $f = \sum c_i \pi_i$, $c_i \in \mathbb{F}_p$:
  $\mathbb{F}_p$-linear combinations of k vectors in $\mathbb{F}_p^n$.

Signature scheme signs k vectors $\mathbf{v}_i \in \mathbb{F}_p^n$ and can authenticate any $\mathbb{F}_p$-linear combination of the $\mathbf{v}_i$.

Same functionality as *network coding* signatures [BFKW09,GKKR10], except p can be small (even $p = 2$).

Security

What does it mean to forge a homomorphic signature?

- Forgery is a valid signature on (τ, m^*, f) with $m^* \neq f(\text{messages with tag } \tau)$.

What does it mean to forge a homomorphic signature?

- Forgery is a valid signature on (τ, m^*, f) with $m^* \neq f(\text{messages with tag } \tau)$.

Chall.

sk

Adversary

Security

What does it mean to forge a homomorphic signature?

- Forgery is a valid signature on (τ, m^*, f) with $m^* \neq f(\text{messages with tag } \tau)$.

$\xrightarrow{pk}$

Chall.

sk

Adversary

What does it mean to forge a homomorphic signature?

- Forgery is a valid signature on (τ, m^*, f) with $m^* \neq f(\text{messages with tag } \tau)$.

Chall.

sk

$\xrightarrow{\quad pk \quad}$

$\xleftarrow{\quad \text{data } m_1, \ldots, m_k \quad}$

Adversary

What does it mean to forge a homomorphic signature?

- Forgery is a valid signature on (τ, m^*, f) with $m^* \neq f(\text{messages with tag } \tau)$.

Chall.

sk

$\xrightarrow{pk}$

$\xleftarrow{\text{data } m_1, \ldots, m_k}$

$\xrightarrow{\text{tag } \tau,\ \text{sigs } \sigma_1, \ldots, \sigma_k}$

Adversary

Security

What does it mean to forge a homomorphic signature?

- Forgery is a valid signature on (τ, m^*, f) with $m^* \neq f(\text{messages with tag } \tau)$.

Chall.

sk

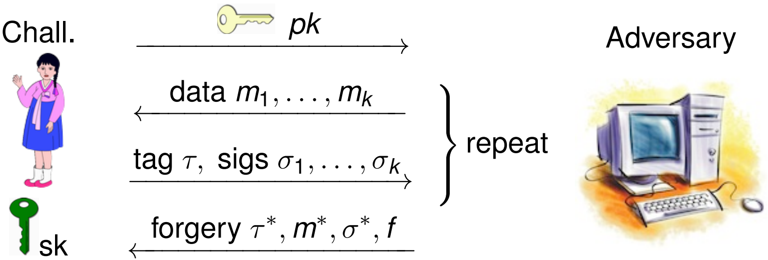
$$\xrightarrow{\quad pk \quad}$$

$$\left.\begin{array}{c} \xleftarrow{\text{data } m_1, \ldots, m_k} \\ \xrightarrow{\text{tag } \tau,\ \text{sigs } \sigma_1, \ldots, \sigma_k} \end{array}\right\} \text{repeat}$$

Adversary

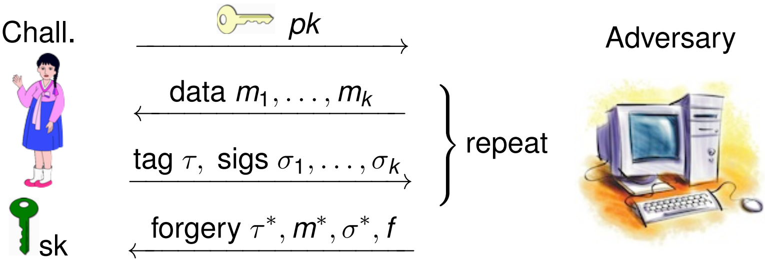
What does it mean to forge a homomorphic signature?

- Forgery is a valid signature on (τ, m^*, f) with $m^* \neq f(\text{messages with tag } \tau)$.



What does it mean to forge a homomorphic signature?

- Forgery is a valid signature on (τ, m^*, f) with $m^* \neq f(\text{messages with tag } \tau)$.



Adversary wins if f admissible, σ^* verifies for (τ^*, m^*, f), and

1. τ^* not obtained in response to a query, or
2. $\tau^* = \tau$ for query $(m_1, \ldots, m_k)$, and $m^* \neq f(m_1, \ldots, m_k)$.

Security Theorem

Theorem

An adversary that wins the security game (in the random oracle model) can be used to compute a short nonzero vector in Λ_2.

Security Theorem

Theorem

An adversary that wins the security game (in the random oracle model) can be used to compute a short nonzero vector in Λ_2.

To implement system securely:
Choose Λ_2 such that finding short vectors in Λ_2 is hard!

- $\Lambda_q^{\perp}(\mathbf{A})$ has this property [MR04,GPV08].

Security Theorem

Theorem

An adversary that wins the security game (in the random oracle model) can be used to compute a short nonzero vector in Λ_2.

To implement system securely:
Choose Λ_2 such that finding short vectors in Λ_2 is hard!

- $\Lambda_q^{\perp}(\mathbf{A})$ has this property [MR04,GPV08].

Proof outline

1. Given a "challenge" Λ_2, answer signature queries without a basis of $\Lambda_1 \cap \Lambda_2$.
2. Use adversary's forgery to produce a short vector in Λ_2.

Security Proof, Part 1

Proof outline

1. Given a “challenge” Λ_2, answer signature queries without a basis of $\Lambda_1 \cap \Lambda_2$.

Security Proof, Part 1

Proof outline

1. Given a "challenge" Λ_2, answer signature queries without a basis of $\Lambda_1 \cap \Lambda_2$.

- Generate Λ_1 with a short basis.

Proof outline

1. Given a “challenge” Λ_2, answer signature queries without a basis of $\Lambda_1 \cap \Lambda_2$.

- Generate Λ_1 with a short basis.
- Adversary queries $m_1, \ldots, m_k$.

Security Proof, Part 1

Proof outline

1. Given a "challenge" Λ_2, answer signature queries without a basis of $\Lambda_1 \cap \Lambda_2$.

- Generate Λ_1 with a short basis.
- Adversary queries $m_1, \ldots, m_k$.
- Choose random τ and simulate signature σ_i on (τ, m_i, π_i):
  1. Use basis of Λ_1 to compute short vectors $\sigma_i \in \Lambda_1 + m_i$;
  2. Set $\alpha_i := \sigma_i \bmod \Lambda_2 \in \mathbb{Z}^n/\Lambda_2$.
  3. Program random oracle with $H(\tau) := (\alpha_1, \ldots, \alpha_k)$.

Security Proof, Part 1

Proof outline

1. Given a "challenge" Λ_2, answer signature queries without a basis of $\Lambda_1 \cap \Lambda_2$.

- Generate Λ_1 with a short basis.
- Adversary queries $m_1, \ldots, m_k$.
- Choose random τ and simulate signature σ_i on (τ, m_i, π_i):
  1. Use basis of Λ_1 to compute short vectors $\sigma_i \in \Lambda_1 + m_i$;
  2. Set $\alpha_i := \sigma_i \bmod \Lambda_2 \in \mathbb{Z}^n/\Lambda_2$.
  3. Program random oracle with $H(\tau) := (\alpha_1, \ldots, \alpha_k)$.
- For certain parameter choices, α_i are statistically close to uniform in $\mathbb{Z}^n/\Lambda_2$.

Security Proof, Part 1

Proof outline

1. Given a "challenge" Λ_2, answer signature queries without a basis of $\Lambda_1 \cap \Lambda_2$.

- Generate Λ_1 with a short basis.
- Adversary queries $m_1, \ldots, m_k$.
- Choose random τ and simulate signature σ_i on (τ, m_i, π_i):
  1. Use basis of Λ_1 to compute short vectors $\sigma_i \in \Lambda_1 + m_i$;
  2. Set $\alpha_i := \sigma_i \bmod \Lambda_2 \in \mathbb{Z}^n/\Lambda_2$.
  3. Program random oracle with $H(\tau) := (\alpha_1, \ldots, \alpha_k)$.
- For certain parameter choices, α_i are statistically close to uniform in $\mathbb{Z}^n/\Lambda_2$.
- Simulation is indistinguishable from real system.

Security Proof, Part 2

Proof outline

2. Use forgery to produce a short nonzero vector in Λ_2.

Security Proof, Part 2

Proof outline

2. Use forgery to produce a short nonzero vector in Λ_2.

- Adversary outputs forgery $(\tau^*, m^*, \sigma^*, f = \sum c_i \pi_i)$.

Security Proof, Part 2

Proof outline

2. Use forgery to produce a short nonzero vector in Λ_2.

- Adversary outputs forgery $(\tau^*, m^*, \sigma^*, f = \sum c_i \pi_i)$.
- Suppose $\tau^*, \sigma_1, \ldots, \sigma_k$ answer query $m_1, \ldots, m_k$.

Security Proof, Part 2

Proof outline

2. Use forgery to produce a short nonzero vector in Λ_2.

- Adversary outputs forgery $(\tau^*, m^*, \sigma^*, f = \sum c_i \pi_i)$.
- Suppose $\tau^*, \sigma_1, \ldots, \sigma_k$ answer query $m_1, \ldots, m_k$.
- Compute $\sigma := \sum c_i \sigma_i =$ "real" sig on $f(m_1, \ldots, m_k)$.

Security Proof, Part 2

Proof outline

2. Use forgery to produce a short nonzero vector in Λ_2.

- Adversary outputs forgery $(\tau^*, m^*, \sigma^*, f = \sum c_i \pi_i)$.
- Suppose $\tau^*, \sigma_1, \ldots, \sigma_k$ answer query $m_1, \ldots, m_k$.
- Compute $\sigma := \sum c_i \sigma_i =$ "real" sig on $f(m_1, \ldots, m_k)$.
- Validity of forged σ^* means:
  1. $\sigma^* \bmod \Lambda_1 = m^* \neq f(m_1, \ldots, m_k)$
  2. $\sigma^* \bmod \Lambda_2 = \sum c_i \alpha_i$
  3. σ^* is short

Security Proof, Part 2

Proof outline

2. Use forgery to produce a short nonzero vector in Λ_2.

- Adversary outputs forgery $(\tau^*, m^*, \sigma^*, f = \sum c_i \pi_i)$.
- Suppose $\tau^*, \sigma_1, \ldots, \sigma_k$ answer query $m_1, \ldots, m_k$.
- Compute $\sigma := \sum c_i \sigma_i =$ "real" sig on $f(m_1, \ldots, m_k)$.
- Validity of forged σ^* and authenticity of "real" σ means:
    1. $\sigma^* \bmod \Lambda_1 = m^* \neq f(m_1, \ldots, m_k) = \sigma \bmod \Lambda_1$
    2. $\sigma^* \bmod \Lambda_2 = \sum c_i \alpha_i = \sigma \bmod \Lambda_2$
    3. σ^* is short and σ is short

Security Proof, Part 2

Proof outline

2. Use forgery to produce a short nonzero vector in Λ_2.

- Adversary outputs forgery $(\tau^*, m^*, \sigma^*, f = \sum c_i \pi_i)$.
- Suppose $\tau^*, \sigma_1, \ldots, \sigma_k$ answer query $m_1, \ldots, m_k$.
- Compute $\sigma := \sum c_i \sigma_i =$ "real" sig on $f(m_1, \ldots, m_k)$.
- Validity of forged σ^* and authenticity of "real" σ means:
  1. $\sigma^* \bmod \Lambda_1 = m^* \neq f(m_1, \ldots, m_k) = \sigma \bmod \Lambda_1$
  2. $\sigma^* \bmod \Lambda_2 = \sum c_i \alpha_i = \sigma \bmod \Lambda_2$
  3. σ^* is short and σ is short

Conclusion: $\sigma^* - \sigma$ is (1) nonzero, (2) in Λ_2, (3) short.

Security Proof, Part 2

Proof outline

2. Use forgery to produce a short nonzero vector in Λ_2.

- Adversary outputs forgery $(\tau^*, m^*, \sigma^*, f = \sum c_i \pi_i)$.
- Suppose $\tau^*, \sigma_1, \ldots, \sigma_k$ answer query $m_1, \ldots, m_k$.
- Compute $\sigma := \sum c_i \sigma_i =$ "real" sig on $f(m_1, \ldots, m_k)$.
- Validity of forged σ^* and authenticity of "real" σ means:
  1. $\sigma^* \bmod \Lambda_1 = m^* \neq f(m_1, \ldots, m_k) = \sigma \bmod \Lambda_1$
  2. $\sigma^* \bmod \Lambda_2 = \sum c_i \alpha_i = \sigma \bmod \Lambda_2$
  3. σ^* is short and σ is short

Conclusion: $\sigma^* - \sigma$ is (1) nonzero, (2) in Λ_2, (3) short.

- If τ^* not obtained from a query, sign random messages m_i and perform same analysis.

Privacy

Privacy property: derived signature on $f(m_1, \ldots, m_k)$ reveals nothing about $m_1, \ldots, m_k$ beyond value of f.

Privacy

Privacy property: derived signature on $f(m_1, \ldots, m_k)$ reveals nothing about $m_1, \ldots, m_k$ beyond value of f.

Specifically: given data sets

$$\vec{m} = (m_1, \ldots, m_k), \qquad \vec{m}' = (m'_1, \ldots, m'_k)$$

and admissible function f with

$$f(\vec{m}) = f(\vec{m}'),$$

even unbounded adversary cannot distinguish derived signature on $f(\vec{m})$ from derived signature on $f(\vec{m}')$.

Privacy theorem

Theorem

Linearly homomorphic signatures are private.

Privacy theorem

Theorem

Linearly homomorphic signatures are private.

Proof idea

- Distribution of derived signature on $f(\vec{m})$ depends only on f and $f(\vec{m})$, not on $\vec{m}$.
- If $f(\vec{m}) = f(\vec{m}')$, distributions of derived sigs are identical.

Privacy theorem

Theorem

Linearly homomorphic signatures are private.

Proof idea

- Distribution of derived signature on $f(\vec{m})$ depends only on f and $f(\vec{m})$, not on $\vec{m}$.
- If $f(\vec{m}) = f(\vec{m}')$, distributions of derived sigs are identical.

Key technical fact [BF11]: distribution of linear combination of discrete Gaussian samples is also discrete Gaussian.

- Sigs on m_i sampled from discrete Gaussian distribution, derived sigs are linear combinations.

Polynomially Homomorphic Signatures from Ideal Lattices

Extending the system

Linearly homomorphic scheme: messages in $\mathbb{Z}^n/\Lambda_1$, functions "encoded" in $\mathbb{Z}^n/\Lambda_2$, signatures are short vectors in $\mathbb{Z}^n$.

$\phi_i \colon \mathbb{Z}^n \to \mathbb{Z}^n/\Lambda_i$ given by $\mathbf{v} \mapsto (\mathbf{v} \bmod \Lambda_i)$ is a linear map, so we can add either before or after applying ϕ_i.

Extending the system

Linearly homomorphic scheme: messages in $\mathbb{Z}^n/\Lambda_1$, functions "encoded" in $\mathbb{Z}^n/\Lambda_2$, signatures are short vectors in $\mathbb{Z}^n$.

$\phi_i \colon \mathbb{Z}^n \to \mathbb{Z}^n/\Lambda_i$ given by $\mathbf{v} \mapsto (\mathbf{v} \bmod \Lambda_i)$ is a linear map, so we can add either before or after applying ϕ_i.

New idea: what if $\mathbb{Z}^n$ has a ring structure and Λ_1, Λ_2 are ideals?

Then ϕ is a ring homomorphism, so we can add or multiply either before or after applying ϕ.

Extending the system

Linearly homomorphic scheme: messages in $\mathbb{Z}^n/\Lambda_1$, functions "encoded" in $\mathbb{Z}^n/\Lambda_2$, signatures are short vectors in $\mathbb{Z}^n$.

$\phi_i\colon \mathbb{Z}^n \to \mathbb{Z}^n/\Lambda_i$ given by $\mathbf{v} \mapsto (\mathbf{v} \bmod \Lambda_i)$ is a linear map, so we can add either before or after applying ϕ_i.

New idea: what if $\mathbb{Z}^n$ has a ring structure and Λ_1, Λ_2 are ideals?

Then ϕ is a ring homomorphism, so we can add or multiply either before or after applying ϕ.

- Can authenticate polynomial functions on messages.

Setup for polynomial system [G09]

Fix monic, irreducible $F(x) \in \mathbb{Z}[x]$ of degree n.

- $R := \mathbb{Z}[x]/(F(x))$ gives a ring structure on $\mathbb{Z}^n$:

$$\begin{array}{ccc} \text{(coordinates of vectors)} & \leftrightarrow & \text{(coefficients of polynomials mod } F\text{)} \\ (a_0, \ldots, a_{n-1}) & \leftrightarrow & a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} \end{array}$$

Setup for polynomial system [G09]

Fix monic, irreducible $F(x) \in \mathbb{Z}[x]$ of degree n.

- $R := \mathbb{Z}[x]/(F(x))$ gives a ring structure on $\mathbb{Z}^n$:

$$\begin{array}{ccc} \text{(coordinates of vectors)} & \leftrightarrow & \text{(coefficients of polynomials mod } F\text{)} \\ (a_0, \ldots, a_{n-1}) & \leftrightarrow & a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} \end{array}$$

$\Lambda_1 =$ prime ideal $\mathfrak{p} \subset R$ of norm p.

- Message space is $R/\mathfrak{p} = \mathbb{F}_p$.
- Admissible functions are polynomials $f \in \mathbb{F}_p[x_1, \ldots, x_k]$ with small coefficients.

Setup for polynomial system [G09]

Fix monic, irreducible $F(x) \in \mathbb{Z}[x]$ of degree n.

- $R := \mathbb{Z}[x]/(F(x))$ gives a ring structure on $\mathbb{Z}^n$:

$$\begin{array}{ccc} \text{(coordinates of vectors)} & \leftrightarrow & \text{(coefficients of polynomials mod } F\text{)} \\ (a_0, \ldots, a_{n-1}) & \leftrightarrow & a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} \end{array}$$

$\Lambda_1 =$ prime ideal $\mathfrak{p} \subset R$ of norm p.

- Message space is $R/\mathfrak{p} = \mathbb{F}_p$.
- Admissible functions are polynomials $f \in \mathbb{F}_p[x_1, \ldots, x_k]$ with small coefficients.

$\Lambda_2 =$ prime ideal $\mathfrak{q}$; polynomials “encoded” in $R/\mathfrak{q} = \mathbb{F}_q$:

- Hash function $H\colon \{0,1\}^* \to \mathbb{F}_q^k$ maps $\tau \mapsto (\alpha_1, \ldots, \alpha_k)$.
- “Encode” f by $\omega(f) := f(\alpha_1, \ldots, \alpha_k)$.
  (think of coefficients of f as small integers).

The polynomial system, concretely

- KeyGen(n):
  - $F(x) \in \mathbb{Z}[x]$ degree n
    $\Rightarrow$ ring structure on $\mathbb{Z}^n \cong R := \mathbb{Z}[x]/(F(x))$.
  - pk = prime ideals $\mathfrak{p}, \mathfrak{q} \subset R$, Gaussian parameter β
  - sk = short basis of $\mathfrak{p} \cap \mathfrak{q} = \mathfrak{p} \cdot \mathfrak{q}$
  - $H\colon \{0,1\}^* \to \mathbb{F}_q^k, \quad H(\tau) = (\alpha_1, \ldots, \alpha_k)$.

The polynomial system, concretely

- KeyGen(n):
  - $F(x) \in \mathbb{Z}[x]$ degree n
    $\Rightarrow$ ring structure on $\mathbb{Z}^n \cong R := \mathbb{Z}[x]/(F(x))$.
  - pk = prime ideals $\mathfrak{p}, \mathfrak{q} \subset R$, Gaussian parameter β
  - sk = short basis of $\mathfrak{p} \cap \mathfrak{q} = \mathfrak{p} \cdot \mathfrak{q}$
  - $H\colon \{0,1\}^* \to \mathbb{F}_q^k, \quad H(\tau) = (\alpha_1, \ldots, \alpha_k)$.
- Sign(τ, m_i, x_i):
  - Compute short element σ_i in $\mathfrak{p} \cdot \mathfrak{q} + \text{CRT}(m_i, \alpha_i)$.

The polynomial system, concretely

- KeyGen(n):
  - $F(x) \in \mathbb{Z}[x]$ degree n
    $\Rightarrow$ ring structure on $\mathbb{Z}^n \cong R := \mathbb{Z}[x]/(F(x))$.
  - pk = prime ideals $\mathfrak{p}, \mathfrak{q} \subset R$, Gaussian parameter β
  - sk = short basis of $\mathfrak{p} \cap \mathfrak{q} = \mathfrak{p} \cdot \mathfrak{q}$
  - $H\colon \{0,1\}^* \to \mathbb{F}_q^k, \quad H(\tau) = (\alpha_1, \ldots, \alpha_k)$.
- Sign(τ, m_i, x_i):
  - Compute short element σ_i in $\mathfrak{p} \cdot \mathfrak{q} + \mathrm{CRT}(m_i, \alpha_i)$.
- Evaluate($f, (\sigma_1, \ldots, \sigma_k)$):
  - Output $\sigma = f(\sigma_1, \ldots, \sigma_k) \in R$

The polynomial system, concretely

- KeyGen(n):
  - $F(x) \in \mathbb{Z}[x]$ degree n
    $\Rightarrow$ ring structure on $\mathbb{Z}^n \cong R := \mathbb{Z}[x]/(F(x))$.
  - pk = prime ideals $\mathfrak{p}, \mathfrak{q} \subset R$, Gaussian parameter β
  - sk = short basis of $\mathfrak{p} \cap \mathfrak{q} = \mathfrak{p} \cdot \mathfrak{q}$
  - $H\colon \{0,1\}^* \to \mathbb{F}_q^k, \quad H(\tau) = (\alpha_1, \ldots, \alpha_k)$.
- Sign(τ, m_i, x_i):
  - Compute short element σ_i in $\mathfrak{p} \cdot \mathfrak{q} + \text{CRT}(m_i, \alpha_i)$.
- Evaluate($f, (\sigma_1, \ldots, \sigma_k)$):
  - Output $\sigma = f(\sigma_1, \ldots, \sigma_k) \in R$
- Verify(τ, σ, m, f): Accept if
  1. $\sigma \bmod \mathfrak{p} = m$,
  2. $\sigma \bmod \mathfrak{q} = f(\alpha_1, \ldots, \alpha_k)$,
  3. σ sufficiently short

The polynomial system, concretely

- KeyGen(n):
  - $F(x) \in \mathbb{Z}[x]$ degree n
    $\Rightarrow$ ring structure on $\mathbb{Z}^n \cong R := \mathbb{Z}[x]/(F(x))$.
  - pk = prime ideals $\mathfrak{p}, \mathfrak{q} \subset R$, Gaussian parameter β
  - sk = short basis of $\mathfrak{p} \cap \mathfrak{q} = \mathfrak{p} \cdot \mathfrak{q}$ — how to generate?
  - $H\colon \{0,1\}^* \to \mathbb{F}_q^k, \quad H(\tau) = (\alpha_1, \ldots, \alpha_k)$.
- Sign(τ, m_i, x_i):
  - Compute short element σ_i in $\mathfrak{p} \cdot \mathfrak{q} + \mathrm{CRT}(m_i, \alpha_i)$.
- Evaluate($f, (\sigma_1, \ldots, \sigma_k)$):
  - Output $\sigma = f(\sigma_1, \ldots, \sigma_k) \in R$ — why is this short?
- Verify(τ, σ, m, f): Accept if
  1. $\sigma \bmod \mathfrak{p} = m$,
  2. $\sigma \bmod \mathfrak{q} = f(\alpha_1, \ldots, \alpha_k)$,
  3. σ sufficiently short — how short?

Products of short elements

Why is a product of short elements of R short?

Products of short elements

Why is a product of short elements of R short?

[G09,G10]: parameter γ_F measures how much multiplication in R increases length:

$$\gamma_F := \sup_{u,v \in R} \frac{\|u \cdot v\|}{\|u\| \cdot \|v\|}.$$

- Product of d elements of length $< \beta$ has length $< \gamma_F^{d-1} \beta^d$.
- If $\beta, \gamma_F \in \text{poly}(n)$ and $d = O(1)$, then this is still considered "short".

Products of short elements

Why is a product of short elements of R short?

[G09,G10]: parameter γ_F measures how much multiplication in R increases length:

$$\gamma_F := \sup_{u,v \in R} \frac{\|u \cdot v\|}{\|u\| \cdot \|v\|}.$$

- Product of d elements of length $< \beta$ has length $< \gamma_F^{d-1}\beta^d$.
- If $\beta, \gamma_F \in \text{poly}(n)$ and $d = O(1)$, then this is still considered "short".

Lots of $F(x)$ have small γ_F:

- e.g., cyclotomic polynomials $\Phi_\ell(x)$, ℓ prime or $\ell = 2^a 3^b$.

Generating ideals with a short basis

How to generate $\mathfrak{p}, \mathfrak{q}$ with short basis of $\mathfrak{p} \cdot \mathfrak{q}$?

Generating ideals with a short basis

How to generate $\mathfrak{p}, \mathfrak{q}$ with short basis of $\mathfrak{p} \cdot \mathfrak{q}$?

- Smart-Vercauteren: choose a random short $u \in R$, repeat until $u \cdot R$ is a prime ideal (lattice) $\mathfrak{p}$.

Generating ideals with a short basis

How to generate $\mathfrak{p}, \mathfrak{q}$ with short basis of $\mathfrak{p} \cdot \mathfrak{q}$?

- Smart-Vercauteren: choose a random short $u \in R$, repeat until $u \cdot R$ is a prime ideal (lattice) $\mathfrak{p}$.
- Repeat to get a second prime ideal $\mathfrak{q} = v \cdot R$.

Generating ideals with a short basis

How to generate $\mathfrak{p}, \mathfrak{q}$ with short basis of $\mathfrak{p} \cdot \mathfrak{q}$?

- Smart-Vercauteren: choose a random short $u \in R$, repeat until $u \cdot R$ is a prime ideal (lattice) $\mathfrak{p}$.
- Repeat to get a second prime ideal $\mathfrak{q} = v \cdot R$.
- $uv \cdot R = \mathfrak{p} \cdot \mathfrak{q}$, and

$$\mathbf{B} := \{uv,\ uv \cdot x,\ uv \cdot x^2, \ldots,\ uv \cdot x^{n-1}\}.$$

spans $\mathfrak{p} \cdot \mathfrak{q}$ and consists of short elements:

$$\|uv \cdot x^i\| \leq \|u\| \cdot \|v\| \cdot \gamma_F^2.$$

Signature length

How short are derived signatures?

Signature length

How short are derived signatures?

Define admissible function set $\mathcal{F}$ to be polynomials in $\mathbb{F}_p[x_1, \ldots, x_k]$ of degree $\leq d$ with coefficients in $[-y, y]$.

Signature length

How short are derived signatures?

Define admissible function set $\mathcal{F}$ to be polynomials in $\mathbb{F}_p[x_1, \ldots, x_k]$ of degree $\leq d$ with coefficients in $[-y, y]$.

Operation	Length expansion
Evaluate degree-d monomial	$\ell \mapsto \ell^d \cdot \gamma_F^{d-1}$
Multiply by coefficient in $[-y, y]$	$\ell \mapsto \ell \cdot y$
Sum of m monomials of length ℓ	$\ell \mapsto \ell \cdot m$

How short are derived signatures?

Define admissible function set $\mathcal{F}$ to be polynomials in $\mathbb{F}_p[x_1, \ldots, x_k]$ of degree $\leq d$ with coefficients in $[-y, y]$.

Operation	Length expansion
Evaluate degree-d monomial	$\ell \mapsto \ell^d \cdot \gamma_F^{d-1}$
Multiply by coefficient in $[-y, y]$	$\ell \mapsto \ell \cdot y$
Sum of m monomials of length ℓ	$\ell \mapsto \ell \cdot m$

- Signatures on original messages m_i have length $< \beta$
  $\Rightarrow$ signature on $f(m_1, \ldots, m_k)$ has length $< \beta^d \cdot \gamma_F^{d-1} \cdot y \cdot \binom{k+d}{d}$.

Signature length

How short are derived signatures?

Define admissible function set $\mathcal{F}$ to be polynomials in $\mathbb{F}_p[x_1, \ldots, x_k]$ of degree $\leq d$ with coefficients in $[-y, y]$.

Operation	Length expansion
Evaluate degree-d monomial	$\ell \mapsto \ell^d \cdot \gamma_F^{d-1}$
Multiply by coefficient in $[-y, y]$	$\ell \mapsto \ell \cdot y$
Sum of m monomials of length ℓ	$\ell \mapsto \ell \cdot m$

- Signatures on original messages m_i have length $< \beta$
  $\Rightarrow$ signature on $f(m_1, \ldots, m_k)$ has length $< \beta^d \cdot \gamma_F^{d-1} \cdot y \cdot \binom{k+d}{d}$.
- If $\beta, \gamma_F, k, y \in \text{poly}(n)$ and $d = O(1)$, then derived signature length is poly(n). (p is exponential in n)

Signature length

How short are derived signatures?

Define admissible function set $\mathcal{F}$ to be polynomials in $\mathbb{F}_p[x_1, \ldots, x_k]$ of degree $\leq d$ with coefficients in $[-y, y]$.

Operation	Length expansion
Evaluate degree-d monomial	$\ell \mapsto \ell^d \cdot \gamma_F^{d-1}$
Multiply by coefficient in $[-y, y]$	$\ell \mapsto \ell \cdot y$
Sum of m monomials of length ℓ	$\ell \mapsto \ell \cdot m$

- Signatures on original messages m_i have length $< \beta$
  $\Rightarrow$ signature on $f(m_1, \ldots, m_k)$ has length $< \beta^d \cdot \gamma_F^{d-1} \cdot y \cdot \binom{k+d}{d}$.
- If $\beta, \gamma_F, k, y \in \text{poly}(n)$ and $d = O(1)$, then derived signature length is $\text{poly}(n)$. (p is exponential in n)
- For fixed n, bit length of derived signatures is linear in d, logarithmic in k.

What is security based on?

Security of polynomial scheme

What is security based on?

Analysis of linearly homomorphic scheme also applies here:

Theorem

An adversary that wins the security game (in the random oracle model) can be used to compute a short nonzero element of $\mathfrak{q}$.

Security of polynomial scheme

What is security based on?

Analysis of linearly homomorphic scheme also applies here:

Theorem

An adversary that wins the security game (in the random oracle model) can be used to compute a short nonzero element of $\mathfrak{q}$.

- $\mathfrak{q}$ is a principal prime ideal.
  - Producing a short generator of arbitrary principal $\mathfrak{q}$ is a classical problem in algorithmic number theory.

Security of polynomial scheme

What is security based on?

Analysis of linearly homomorphic scheme also applies here:

Theorem

An adversary that wins the security game (in the random oracle model) can be used to compute a short nonzero element of $\mathfrak{q}$.

- $\mathfrak{q}$ is a principal prime ideal.
  - Producing a short generator of arbitrary principal $\mathfrak{q}$ is a classical problem in algorithmic number theory.
- Distribution of Smart-Vercauteren $\mathfrak{q}$ not well understood.
  - Want $\mathfrak{q}$ in distribution that admits a worst-case reduction.

Open questions

1. Construct private polynomially homomorphic signatures. (i.e., that leak no information about original messages)
   - Linearly homomorphic signatures are private.
   - Current polynomial construction is not private.

Open questions

1. Construct private polynomially homomorphic signatures. (i.e., that leak no information about original messages)
   - Linearly homomorphic signatures are private.
   - Current polynomial construction is not private.

2. Remove random oracle from security proof.
   - Work in progress.

Open questions

1. Construct private polynomially homomorphic signatures. (i.e., that leak no information about original messages)
   - Linearly homomorphic signatures are private.
   - Current polynomial construction is not private.
2. Remove random oracle from security proof.
   - Work in progress.
3. Reduce security to worst-case problems on ideal lattices.
   - Achieved for linear scheme.
   - Achieve for polynomial scheme using Gentry's techniques?

Open questions

1. Construct private polynomially homomorphic signatures. (i.e., that leak no information about original messages)
   - Linearly homomorphic signatures are private.
   - Current polynomial construction is not private.

2. Remove random oracle from security proof.
   - Work in progress.

3. Reduce security to worst-case problems on ideal lattices.
   - Achieved for linear scheme.
   - Achieve for polynomial scheme using Gentry's techniques?

4. Fully homomorphic signatures!
   - Adapt "bootstrapping" approach???

Open questions

1. Construct private polynomially homomorphic signatures. (i.e., that leak no information about original messages)
   - Linearly homomorphic signatures are private.
   - Current polynomial construction is not private.
2. Remove random oracle from security proof.
   - Work in progress.
3. Reduce security to worst-case problems on ideal lattices.
   - Achieved for linear scheme.
   - Achieve for polynomial scheme using Gentry's techniques?
4. Fully homomorphic signatures!
   - Adapt "bootstrapping" approach???

Thank you!

Thanks also to Chris Peikert for help with graphics.