Blind Signatures with flying colors

Olivier Blazy

XLim, Université de Limoges

Feb 2014

1 General Remarks

2 Building blocks

3 Non-Interactive Proofs of Knowledge

4 Interactive Implicit Proofs

5 Can we do better?

1 General Remarks

2 Building blocks

3 Non-Interactive Proofs of Knowledge

4 Interactive Implicit Proofs

5 Can we do better?

1. General Remarks
2. Building blocks
3. Non-Interactive Proofs of Knowledge
4. Interactive Implicit Proofs
5. Can we do better?

1. General Remarks
2. Building blocks
3. Non-Interactive Proofs of Knowledge
4. Interactive Implicit Proofs
5. Can we do better?

1 General Remarks

2 Building blocks

3 Non-Interactive Proofs of Knowledge

4 Interactive Implicit Proofs

5 Can we do better?

Electronic Voting

For dessert, we let people vote

- ✓ Chocolate Cake
- ✓ Cheese Cake
- ✓ Fruit Salad
- ✓ Brussels Sprout

After collection, we count the number of ballots:

Chocolate Cake	123
Cheese Cake	79
Fruit Salad	42
Brussels sprout	1

Authentication

- Only people authorized to vote should be able to vote
- People should be able to vote only once

Anonymity

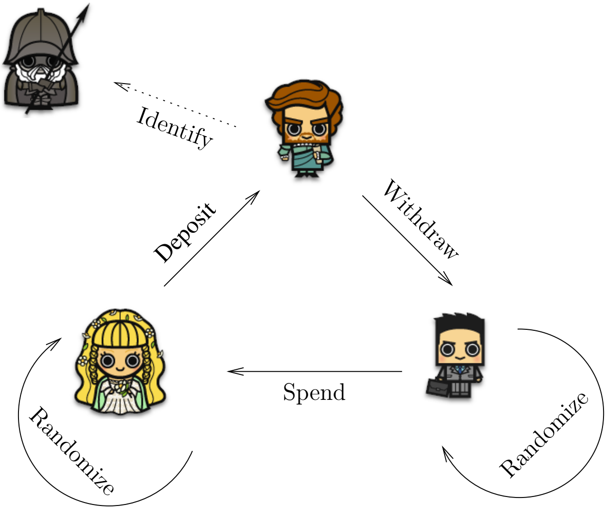
- Votes and voters should be anonymous

△ Receipt freeness

Homomorphic Encryption and Signature approach

- The voter generates his vote v.
- The voter encrypts v to the server as c.
- The voter signs c and outputs σ.
- (c, σ) is a ballot unique per voter, and anonymous.
- Counting: granted homomorphic encryption $C = \prod c$.
- The server decrypts C.

Electronic Cash



Protocol

- Withdrawal: A user get a coin c from the bank
- Spending: A user pays a shop with the coin c
- Deposit: The shop gives the coin c back to the bank

Electronic Coins — Chaum 81

Expected properties

✓ *Unforgeability* $\rightsquigarrow$ Coins are signed by the bank

✓ *No Double-Spending* $\rightsquigarrow$ Each coin is unique

✓ *Anonymity* $\rightsquigarrow$ Blind Signature

Definition (Blind Signature)

A blind signature allows a user to get a message m signed by an authority into σ so that the authority *even powerful* cannot recognize later the pair (m, σ).

RSA-Based Blind Signature

The easiest way for blind signatures, is to blind the message:
To get an FDH-RSA signature on m under RSA public key (n, e),

- The user computes a blind version of the hash value:

$$M = H(m) \text{ and } M' = M \cdot r^e \bmod n$$

- The signer signs M' into $\sigma' = M'^d$
- The user recovers $\sigma = \sigma'/r$

$\rightarrow$ Proven under the One-More RSA Assumption in 2001
$\rightarrow$ Perfectly Blind Signature

Round-Optimal Blind Signature — Fischlin 06

- The user encrypts his message m in c.
- The signer then signs c in σ.
- The user verifies σ.
- He then encrypts σ and c into $\mathcal{C}_\sigma$ and $\mathcal{C}$ and generates a proof π.
- π: $\mathcal{C}_\sigma$ is an encryption of a signature over the ciphertext c encrypted in $\mathcal{C}$, and this c is indeed an encryption of m.
- Anyone can then use $\mathcal{C}, \mathcal{C}_\sigma, \pi$ to check the validity of the signature.

Vote

- A user should be able to encrypt a ballot.
- He should be able to sign this encryption.
- Receiving this vote, one should be able to randomize for *Receipt-Freeness*.

E-Cash

- A user should be able to encrypt a token
- The bank should be able to sign it providing *Unforgeability*
- This signature should now be able to be randomized to provide *Anonymity*

Our Solution

- Same underlying requirements;
- Advance security notions in both schemes requires to extract some kind of signature on the associated plaintext;
- General Framework for Signature on Randomizable Ciphertexts;
- $\rightsquigarrow$ Revisited Waters, Commutative encryption / signature.

1 General Remarks

2 Building blocks
- Bilinear groups aka Pairing-friendly environments
- Commitment / Encryption
- Signatures
- Security hypotheses

3 Non-Interactive Proofs of Knowledge

4 Interactive Implicit Proofs

5 Can we do better?

Asymmetric bilinear structure

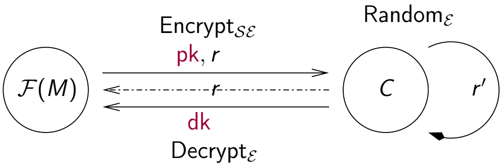
$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ bilinear structure:

- $\mathbb{G}_1, \mathbb{G}_2$, $\mathbb{G}_T$ multiplicative groups of order p
  - $p =$ **prime integer**
- $\langle g_* \rangle = \mathbb{G}_*$
- $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$
  - $\langle e(g_1, g_2) \rangle = \mathbb{G}_T$
  - $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$, $a, b \in \mathbb{Z}$
- $\left.\begin{array}{l}\text{deciding group membership,}\\ \text{group operations,}\\ \text{bilinear map}\end{array}\right\}$ efficiently computable.

Definition (Encryption Scheme)

$\mathcal{E} = (\mathsf{Setup}, \mathsf{EKeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$:

- $\mathsf{Setup}(1^{\mathfrak{K}})$: param;
- EKeyGen(param): public *encryption* key pk, private *decryption* key dk;
- $\mathsf{Encrypt}(\mathsf{pk}, m; r)$: ciphertext c on $m \in \mathcal{M}$ and pk;
- $\mathsf{Decrypt}(\mathsf{dk}, c)$: decrypts c under dk.



Indistinguishability:
Given M_0, M_1, it should be hard to guess which one is encrypted in C.

Definition (ElGamal Encryption) (84)

- Setup($1^{\mathfrak{K}}$): Generates a multiplicative group $(p, \mathbb{G}, g)$.
- EKeyGen$_{\mathcal{E}}$(param): dk $= \mu \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, and pk $= (X_1 = g^\mu)$.
- Encrypt(pk $= X_1, M; \alpha$): For M, and random $\alpha \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, $\mathcal{C} = \left(c_1 = X_1^\alpha, c_2 = g^\alpha \cdot M\right)$.
- Decrypt(dk $= (\mu), \mathcal{C} = (c_1, c_2)$): Computes $M = c_2/(c_1^{1/\mu})$.

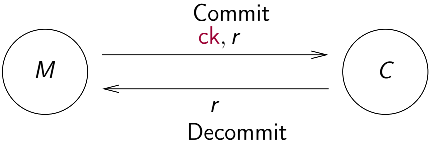
Randomization

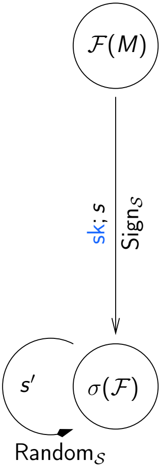
Random(pk, $\mathcal{C}; r) : \mathcal{C}' = \left(c_1 X_1^r, c_2 g^r\right) = \left(X_1^{\alpha+r}, g^{\alpha+r} \cdot M\right)$

Definition (Commitment Scheme)

$\mathcal{E} = (\text{Setup}, \text{Commit}, \text{Decommit})$:

- $\text{Setup}(1^{\mathfrak{K}})$: param, ck;
- Commit(ck, m; r): **c** on the input message $m \in \mathcal{M}$ using $r \stackrel{\$}{\leftarrow} \mathcal{R}$;
- Decommit(**c**, m; w) opens **c** and reveals m, together with w that proves the correct opening.





Definition (Signature Scheme)

$\mathcal{S} = (\mathsf{Setup}, \mathsf{SKeyGen}, \mathsf{Sign}, \mathsf{Verif})$:

- Setup($1^{\mathfrak{K}}$): param;
- SKeyGen(param): public *verification* key vk, private *signing* key sk;
- Sign(sk, m; s): signature σ on m, under sk;
- Verif(vk, m, σ): checks whether σ is valid on m.

Unforgeability:
Given q pairs (m_i, σ_i), it should be hard to output a valid σ on a fresh m.

Definition (Waters Signature) (Wat05)

- $\mathsf{Setup}_{\mathcal{S}}(1^{\mathfrak{K}})$: Generates $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, an extra h, and (u_i) for the Waters function $(\mathcal{F}(m) = u_0 \prod_i u_i^{m_i})$.
- $\mathsf{SKeyGen}_{\mathcal{S}}(\mathsf{param})$: Picks $x \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ and outputs $\mathsf{sk} = h^x$, and $\mathsf{vk} = g^x$;
- $\mathsf{Sign}(\mathsf{sk}, m; s)$: Outputs $\sigma(m) = (\mathsf{sk}\mathcal{F}(m)^s, g^s)$;
- $\mathsf{Verif}(\mathsf{vk}, m, \sigma)$: Checks the validity of σ: $e(g, \sigma_1) \stackrel{?}{=} e(\mathcal{F}(m), \sigma_2) \cdot e(\mathsf{vk}, h)$

Randomization

$\mathsf{Random}(\sigma; r) : \sigma' = \left(\sigma_1 \mathcal{F}(m)^r, \sigma_2 g^r\right) = \left(\mathsf{sk}\mathcal{F}(m)^{r+s}, g^{r+s}\right)$

Definition (DL)

Given $g, h \in \mathbb{G}^2$, it is hard to compute α such that $h = g^\alpha$.

Definition (CDH)

Given $g, g^a, h \in \mathbb{G}^3$, it is hard to compute h^a.

1 General Remarks

2 Building blocks

3 Non-Interactive Proofs of Knowledge
- Groth Sahai methodology
- Signature on Ciphertexts
- Application to other protocols
- Waters Programmability

4 Interactive Implicit Proofs

5 Can we do better?

Groth-Sahai Proof System

- **Pairing product equation (PPE):** for variables $\mathcal{X}_1, \ldots, \mathcal{X}_m \in \mathbb{G}_1$

$$(E) : \prod_{j=1}^{n} e(A_j, \mathcal{Y}_J) \prod_{i=1}^{m} e(\mathcal{X}_i, B_i) \prod_{i=1}^{m} \prod_{j=1}^{n} e(\mathcal{X}_i, \mathcal{Y}_j)^{\gamma_{i,j}} = t_T$$

determined by $A_i \in \mathbb{G}_1, B_i \in \mathbb{G}_2$, $\gamma_{i,j} \in \mathbb{Z}_p$ and $t_T \in \mathbb{G}_T$.

- Groth-Sahai $\rightsquigarrow$ WI proofs that elements that were committed satisfy PPE

Setup($\mathbb{G}$): commitment key **ck**;
Com(**ck**, $X \in \mathbb{G}; \rho$): commitment $\vec{c_X}$ to X;
Prove(**ck**, $(X_i, \rho_i)_{i=1,\ldots,n}$, (E)): proof ϕ;
Verify(**ck**, $\vec{c_{X_i}}$, (E), ϕ): checks whether ϕ is valid.

Groth-Sahai Proof System

- **Pairing product equation (PPE):** for variables $\mathcal{X}_1, \ldots, \mathcal{X}_m \in \mathbb{G}_1$

$$(E) : \prod_{j=1}^{n} e(A_j, \mathcal{Y}_J) \prod_{i=1}^{m} e(\mathcal{X}_i, B_i) \prod_{i=1}^{m} \prod_{j=1}^{n} e(\mathcal{X}_i, \mathcal{Y}_j)^{\gamma_{i,j}} = t_T$$

determined by $A_i \in \mathbb{G}_1, B_i \in \mathbb{G}_2$, $\gamma_{i,j} \in \mathbb{Z}_p$ and $t_T \in \mathbb{G}_T$.

- Groth-Sahai $\rightsquigarrow$ WI proofs that elements that were committed satisfy PPE

Setup($\mathbb{G}$): commitment key **ck**;
Com(**ck**, $X \in \mathbb{G}$; ρ): commitment $\vec{c_X}$ to X;
Prove(**ck**, $(X_i, \rho_i)_{i=1,\ldots,n}$, (E)): proof ϕ;
Verify(**ck**, $\vec{c_{X_i}}$, (E), ϕ): checks whether ϕ is valid.

$$(E) : \prod_{j=1}^{n} e(A_j, \mathcal{Y}_J) \prod_{i=1}^{m} e(\mathcal{X}_i, B_i) \prod_{i=1}^{m} \prod_{j=1}^{n} e(\mathcal{X}_i, \mathcal{Y}_j)^{\gamma_{i,j}} = t_T$$

Assumption	DLin	SXDH
Variables	3	2
PPE	9	(4,4)
Linear	3	2

Verification	$12n + 27$	$5m + 3n + 16$
[ACNS 2010: BFI+]	$3n + 6$	$m + 2n + 8$

Properties:

- correctness
- soundness
- witness-indistinguishability
- randomizability Commitments and proofs are publicly randomizable.

$$(E): \prod_{j=1}^{n} e(A_j, \mathcal{Y}_J) \prod_{i=1}^{m} e(\mathcal{X}_i, B_i) \prod_{i=1}^{m} \prod_{j=1}^{n} e(\mathcal{X}_i, \mathcal{Y}_j)^{\gamma_{i,j}} = t_T$$

Assumption	DLin	SXDH
Variables	3	2
PPE	9	(4,4)
Linear	3	2
Verification	$12n + 27$	$5m + 3n + 16$
[ACNS 2010: BFI+]	$3n + 6$	$m + 2n + 8$

Properties:

- correctness
- soundness
- witness-indistinguishability
- randomizability Commitments and proofs are publicly randomizable.

$$(E): \prod_{j=1}^{n} e(A_j, \mathcal{Y}_J) \prod_{i=1}^{m} e(\mathcal{X}_i, B_i) \prod_{i=1}^{m} \prod_{j=1}^{n} e(\mathcal{X}_i, \mathcal{Y}_j)^{\gamma_{i,j}} = t_T$$

Assumption	DLin	SXDH
Variables	3	2
PPE	9	(4,4)
Linear	3	2
Verification	$12n + 27$	$5m + 3n + 16$
[ACNS 2010: BFI+]	$3n + 6$	$m + 2n + 8$

Properties:

- correctness
- soundness
- witness-indistinguishability
- randomizability Commitments and proofs are publicly randomizable.

Commutative properties

Encrypt

To encrypt a message m:

$$c = (\mathsf{pk}^r, \mathcal{F}(m) \cdot g^r)$$

Commutative properties

Encrypt

To encrypt a message m:

$$c = (\mathsf{pk}^r, \mathcal{F}(m) \cdot g^r)$$

Sign ∘ Encrypt

To sign a valid ciphertext c_1, c_2, c_3, one has simply to produce.

$$\sigma = ({c_1}^s, \mathsf{sk} \cdot {c_2}^s, \mathsf{pk}^s, g^s) \ .$$

Commutative properties

Encrypt

To encrypt a message m:

$$c = (\mathsf{pk}^r, \mathcal{F}(m) \cdot g^r)$$

Sign ∘ Encrypt

To sign a valid ciphertext c_1, c_2, c_3, one has simply to produce.

$$\sigma = ({c_1}^s, \mathsf{sk} \cdot {c_2}^s, \mathsf{pk}^s, g^s) \ .$$

Decrypt ∘ Sign ∘ Encrypt

Using dk.

$$\sigma = (\sigma_2 / \sigma_1^{\mathsf{dk}}, \sigma_4) = (\mathsf{sk} \cdot \mathcal{F}(m)^s, g^s) \ .$$

Definition (Signature on Ciphertexts)

$\mathcal{SE}$ = (Setup, SKeyGen, EKeyGen, Encrypt, Sign, Decrypt, Verif):

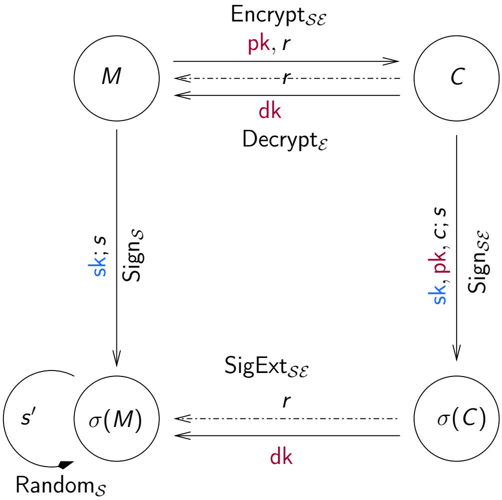
- Setup($1^{\mathfrak{K}}$): param_e, param_s;
- EKeyGen(param_e): pk, dk;
- SKeyGen(param_s): vk, sk;
- Encrypt(pk, vk, m; r): produces c on $m \in \mathcal{M}$ and pk;
- Sign(sk, pk, c; s): produces σ, on the input c under sk;
- Decrypt(dk, vk, c): decrypts c under dk;
- Verif(vk, pk, c, σ): checks whether σ is valid.

Definition (Extractable Randomizable Signature on Ciphertexts)

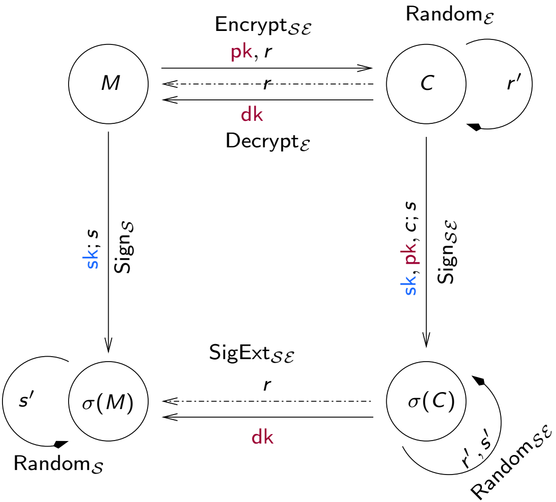
$\mathcal{SE}$=(Setup, SKeyGen, EKeyGen, Encrypt, Sign, Random, Decrypt, Verif, SigExt):

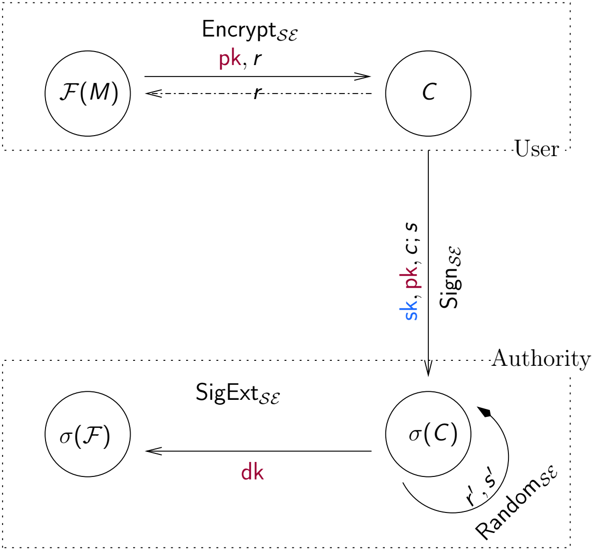
- Random(vk, pk, c, σ; r', s') produces c' and σ' on c', using additional coins;
- SigExt(dk, vk, σ) outputs a signature σ^*.

Randomizable Signature on Ciphertexts [PKC 2011: BFPV]

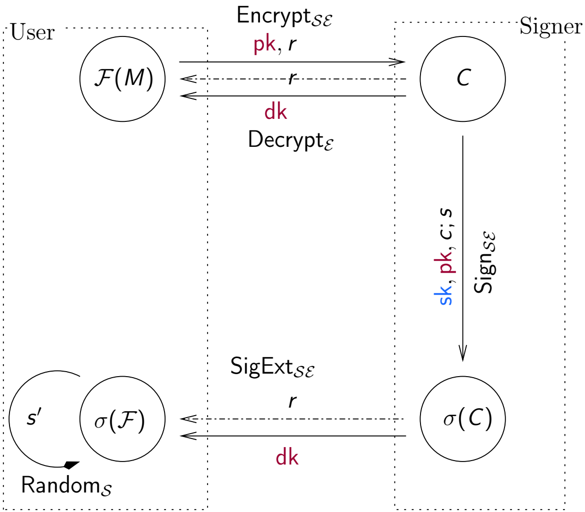


Extractable SRC

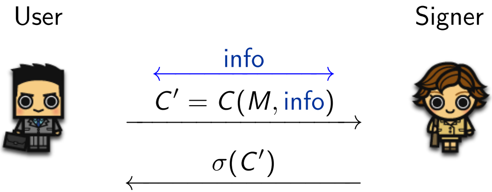



Encrypt$_{\mathcal{SE}}$
pk, r
$\mathcal{F}(M)$
r
C
User
sk, pk, C; s
Sign$_{\mathcal{SE}}$
Authority
SigExt$_{\mathcal{SE}}$
$\sigma(\mathcal{F})$
$\sigma(C)$
dk
r', s'
Random$_{\mathcal{SE}}$

Blind Signature [PKC 2011: BFPV]



Partially-Blind Signature



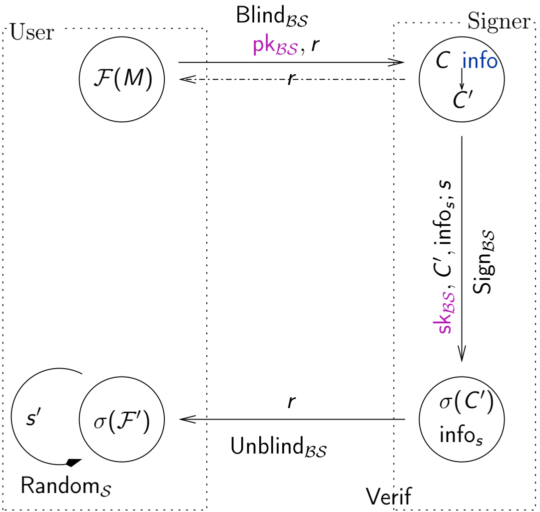
Partially-Blind Signature

User | Signer

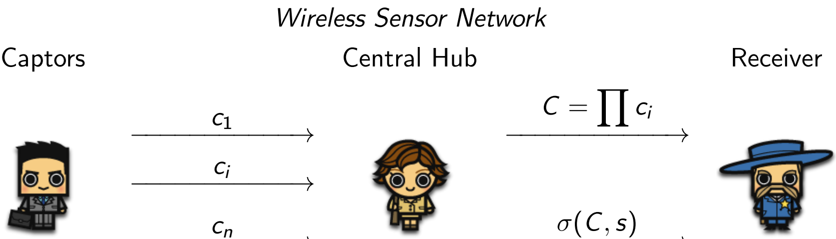
$$C' = C(M, \text{info}) \longrightarrow$$

$$\longleftarrow \sigma(C', \text{info}_s)$$

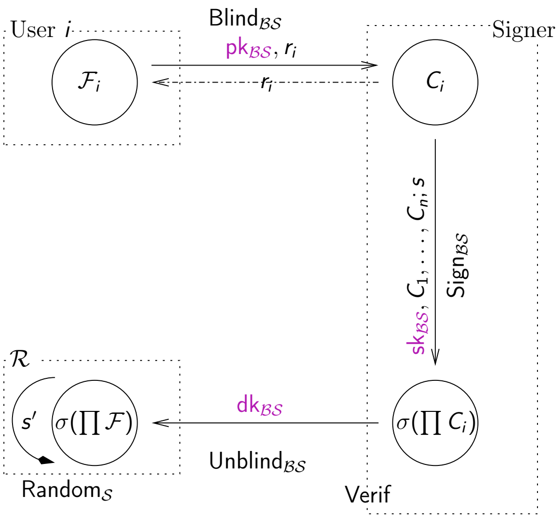
Signer-Friendly Partially Blind Signature [SCN 2012: BPV]



Multi-Source Blind Signatures



Multi-Source Blind Signatures [SCN 2012: BPV]



Two solutions

Different Generators

- Each captor has a disjoint set of generators for the Waters function
- Enormous public key

Two solutions

Different Generators

- Each captor has a disjoint set of generators for the Waters function
- Enormous public key

Two solutions

Different Generators

- Each captor has a disjoint set of generators for the Waters function
- Enormous public key

A single set of generators

- The captors share the same set of generators
- Waters over a non-binary alphabet?

Two solutions

Different Generators

- Each captor has a disjoint set of generators for the Waters function
- Enormous public key

A single set of generators

- The captors share the same set of generators
- Waters over a non-binary alphabet?

Programmability of Waters over a non-binary alphabet

Definition ((m, n)-programmability)

F is (m, n) programmable if given g, h there is an efficient trapdoor producing a_X, b_X such that $F(X) = g^{a_X} h^{b_X}$, and for all X_i, Z_j,
$Pr[a_{X_1} = \cdots = a_{X_m} = 0 \wedge a_{Z_1} \cdot \ldots \cdot a_{Z_n} \neq 0]$ is not negligible.

(1, q)-Programmability of Waters function

Why do we need it: Unforgeabilty, q signing queries, 1 signature to exploit.
$\rightsquigarrow$ Choose independent and uniform elements $(a_i)_{(1,\ldots,\ell)}$ in $\{-1, 0, 1\}$, and random exponents $(b_i)_{(0,\ldots,\ell)}$, and setting $a_0 = -1$.
Then $u_i = g^{a_i} h^{b_i}$.

$$\mathcal{F}(m) = u_0 \prod u_i^{m_i} = g^{\sum_{\delta_i} a_i} h^{\sum_{\delta_i} b_i} = g^{a_m} h^{b_m}.$$

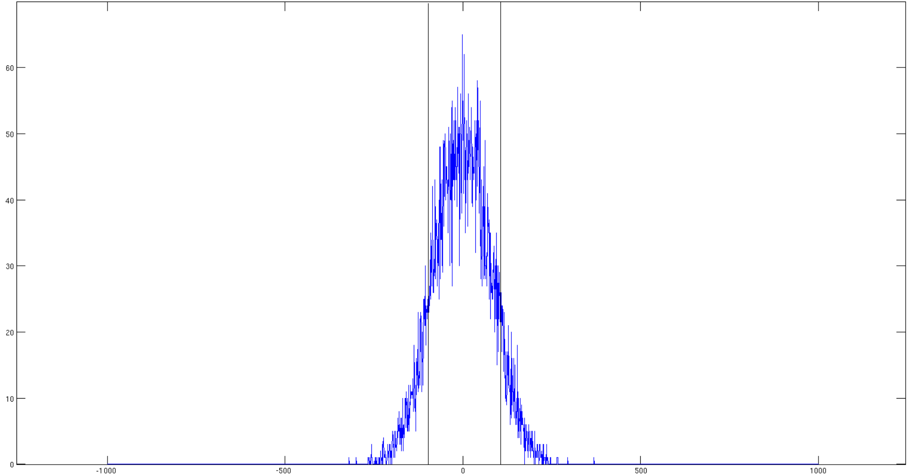
Non $(2,1)$-programmability

Waters over a non-binary alphabet is not $(2,1)$-programmable.

$(1,q)$-programmability

Waters over a polynomial alphabet remains $(1,q)$-programmable.

Sum of random walks on polynomial alphabets



Local Central Limit Theorem $\rightleftharpoons$ Lindeberg Feller

- New primitive: Signature on Randomizable Ciphertexts [PKC 2011: BFPV]
- ✓ One Round Blind Signature [PKC 2011: BFPV]
- ✓ Receipt Free E-Voting [PKC 2011: BFPV]
- ✓ Signer-Friendly Blind Signature [SCN 2012: BPV]
- ✓ Multi-Source Blind Signature [SCN 2012: BPV]

Efficiency

- DLin + CDH : $9\ell + 24$ Group elements.
- SXDH + CDH^+ : $6\ell + 15, 6\ell + 7$ Group elements.

1 General Remarks

2 Building blocks

3 Non-Interactive Proofs of Knowledge

4 Interactive Implicit Proofs
- Motivation
- Smooth Projective Hash Function
- Application

5 Can we do better?

Certification of Public Keys: (NI)ZKPoK

Certification of a public key

Server

$\mathsf{pk} \leftarrow$

$\rightarrow \pi(\mathsf{sk}) \leftarrow$

$\rightarrow \mathsf{Cert}$

User

Certification of Public Keys: (NI)ZKPoK

Certification of a public key

Server

$\mathsf{pk} \leftarrow$

$\rightarrow \pi(\mathsf{sk}) \leftarrow$

$\rightarrow \mathsf{Cert}$

User

Certification of Public Keys: (NI)ZKPoK

Certification of a public key

Server		User
	pk $\leftarrow$ π(sk) $\rightarrow$ Cert	

Certification of Public Keys: (NI)ZKPoK

Certification of a public key

Server

pk ←

π(sk)

→ Cert

User

Certification of Public Keys: (NI)ZKPoK

Certification of a public key

Server

$\mathsf{pk} \leftarrow$

$\pi(\mathsf{sk})$

$\rightarrow \mathsf{Cert}$

User

π can be forwarded

Certification of Public Keys: SPHF [ACP09]

A user can ask for the certification of pk, but if he knows the associated sk only:

With a Smooth Projective Hash Function

$\mathcal{L}$: pk and $C = \mathcal{C}(\mathsf{sk}; r)$ are associated to the same sk

- U sends his pk, and an encryption C of sk;
- A generates the certificate Cert for pk, and sends it, masked by $\mathsf{Hash} = \mathsf{Hash}(\mathsf{hk}; (\mathsf{pk}, C))$;
- U computes $\mathsf{Hash} = \mathsf{ProjHash}(\mathsf{hp}; (\mathsf{pk}, C), r))$, and gets Cert.

Certification of Public Keys: SPHF [ACP09]

A user can ask for the certification of pk, but if he knows the associated sk only:

With a Smooth Projective Hash Function

$\mathcal{L}$: pk and $C = \mathcal{C}(\mathsf{sk}; r)$ are associated to the same sk

- U sends his pk, and an encryption C of sk;
- A generates the certificate Cert for pk, and sends it, masked by $\mathsf{Hash} = \mathsf{Hash}(\mathsf{hk}; (\mathsf{pk}, C))$;
- U computes $\mathsf{Hash} = \mathsf{ProjHash}(\mathsf{hp}; (\mathsf{pk}, C), r))$, and gets Cert.

Implicit proof of knowledge of sk

Smooth Projective Hash Functions [CS02]

Definition [CS02,GL03]

Let $\{H\}$ be a family of functions:

- X, domain of these functions
- L, subset (a language) of this domain

such that, for any point x in L, $H(x)$ can be computed by using

- either a *secret* hashing key hk: $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x)$;
- or a *public* projected key hp: $H'(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$

Public mapping $\mathsf{hk} \mapsto \mathsf{hp} = \mathsf{ProjKG}_L(\mathsf{hk}, x)$

SPHF Properties

For any $x \in X$, $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x)$
For any $x \in L$, $H(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$
w witness that $x \in L, \mathsf{hp} = \mathsf{ProjKG}_L(\mathsf{hk}, x)$

Smoothness

For any $x \notin L$, $H(x)$ and hp are independent

Pseudo-Randomness

For any $x \in L$, $H(x)$ is pseudo-random, without a witness w

The latter property requires L to be a hard-partitioned subset of X.

SPHF Properties

For any $x \in X$, $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x)$
For any $x \in L$, $H(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$
w witness that $x \in L$, $\mathsf{hp} = \mathsf{ProjKG}_L(\mathsf{hk}, x)$

Smoothness

For any $x \notin L$, $H(x)$ and hp are independent

Pseudo-Randomness

For any $x \in L$, $H(x)$ is pseudo-random, without a witness w

The latter property requires L to be a hard-partitioned subset of X.

SPHF Properties

For any $x \in X$, $H(x) = \text{Hash}_L(\text{hk}; x)$
For any $x \in L$, $H(x) = \text{ProjHash}_L(\text{hp}; x, w)$
w witness that $x \in L$, $\text{hp} = \text{ProjKG}_L(\text{hk}, x)$

Smoothness

For any $x \notin L$, $H(x)$ and hp are independent

Pseudo-Randomness

For any $x \in L$, $H(x)$ is pseudo-random, without a witness w

The latter property requires L to be a hard-partitioned subset of X.

SPHF Properties

For any $x \in X$, $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x)$
For any $x \in L$, $H(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$

w witness that $x \in L$, $\mathsf{hp} = \mathsf{ProjKG}_L(\mathsf{hk}, x)$

Smoothness

For any $x \notin L$, $H(x)$ and hp are independent

Pseudo-Randomness

For any $x \in L$, $H(x)$ is pseudo-random, without a witness w

The latter property requires L to be a hard-partitioned subset of X.

Certification of Public Keys: SPHF [ACP09]

Certification of a public key

Server

$$\mathsf{pk}, C = \mathcal{C}(\mathsf{sk}; r) \leftarrow$$
$$\rightarrow P = \mathsf{Cert} \oplus \mathsf{Hash}(\mathsf{hk}; (\mathsf{pk}, C))$$
$$\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, C)$$

User

$$P \oplus \mathsf{ProjHash}(\mathsf{hp}; (\mathsf{pk}, C), r) = \mathsf{Cert}$$

Certification of Public Keys: SPHF [ACP09]

Certification of a public key

Server

$$\mathsf{pk}, C = \mathcal{C}(\mathsf{sk}; r) \leftarrow$$

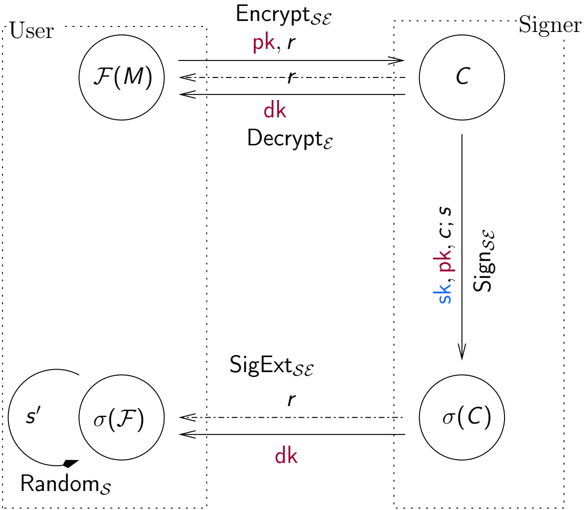
$$\rightarrow P = \mathsf{Cert} \oplus \mathsf{Hash}(\mathsf{hk}; (\mathsf{pk}, C))$$

$$\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, C)$$

User

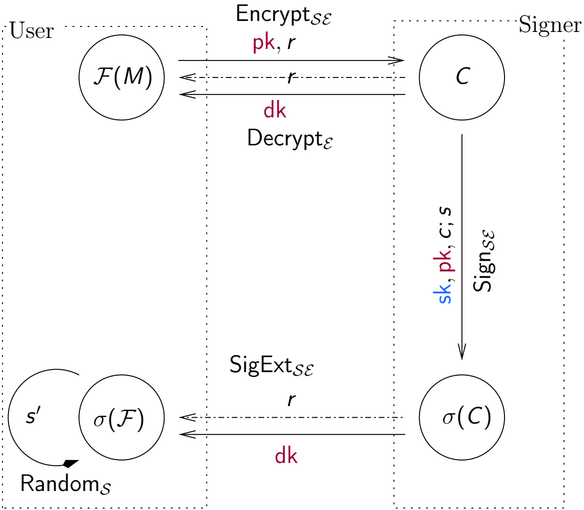
$$P \oplus \mathsf{ProjHash}(\mathsf{hp}; (\mathsf{pk}, C), r) = \mathsf{Cert}$$

Implicit proof of knowledge of sk



Groth Sahai

$6\,\ell + 7, 6\ell + 5$



Groth Sahai

$6\,\ell + 7, 6\ell + 5$

SPHF

$5\,\ell + 6, 1$

Languages

BLin: $\{0, 1\}$,
ELin:
$\{\mathcal{C}(\mathcal{C}(...))\}$.

Smooth Projective Hash Functions $\hat{=}$ *implicit* proofs of knowledge

Various Applications:

Privacy-preserving protocols:

△ Many more Round optimal applications?

Smooth Projective Hash Functions ≙ *implicit* proofs of knowledge

Various Applications:

✓ IND-CCA [CS02]

Privacy-preserving protocols:

△ Many more Round optimal applications?

Smooth Projective Hash Functions ≙ *implicit* proofs of knowledge

Various Applications:

✓ IND-CCA [CS02]

✓ PAKE [GL03]

Privacy-preserving protocols:

△ Many more Round optimal applications?

Smooth Projective Hash Functions $\triangleq$ *implicit* proofs of knowledge

Various Applications:

- ✓ IND-CCA [CS02]
- ✓ PAKE [GL03]
- ✓ Certification of Public Keys [ACP09]

Privacy-preserving protocols:

△ Many more Round optimal applications?

Smooth Projective Hash Functions $\triangleq$ *implicit* proofs of knowledge

Various Applications:

- ✓ IND-CCA [CS02]
- ✓ PAKE [GL03]
- ✓ Certification of Public Keys [ACP09]

Privacy-preserving protocols:

△ Many more Round optimal applications?

Smooth Projective Hash Functions $\hat{=}$ *implicit* proofs of knowledge

Various Applications:

- ✓ IND-CCA [CS02]
- ✓ PAKE [GL03]
- ✓ Certification of Public Keys [ACP09]

Privacy-preserving protocols:

- ✓ Blind signatures [TCC 2012: BPV]

△ Many more Round optimal applications?

Smooth Projective Hash Functions ≙ *implicit* proofs of knowledge

Various Applications:

- ✓ IND-CCA [CS02]
- ✓ PAKE [GL03]
- ✓ Certification of Public Keys [ACP09]

Privacy-preserving protocols:

- ✓ Blind signatures [TCC 2012: BPV]
- ✓ Oblivious Signature-Based Envelope [TCC 2012: BPV]

△ Many more Round optimal applications?

Smooth Projective Hash Functions ≙ *implicit* proofs of knowledge

Various Applications:

- ✓ IND-CCA [CS02]
- ✓ PAKE [GL03]
- ✓ Certification of Public Keys [ACP09]

Privacy-preserving protocols:

- ✓ Blind signatures [TCC 2012: BPV]
- ✓ Oblivious Signature-Based Envelope [TCC 2012: BPV]
- ✓ (v)-PAKE, LAKE, Secret Handshakes [PKC/Crypto 2013: BBCPV]

△ Many more Round optimal applications?

Smooth Projective Hash Functions $\triangleq$ *implicit* proofs of knowledge

Various Applications:

- ✓ IND-CCA [CS02]
- ✓ PAKE [GL03]
- ✓ Certification of Public Keys [ACP09]

Privacy-preserving protocols:

- ✓ Blind signatures [TCC 2012: BPV]
- ✓ Oblivious Signature-Based Envelope [TCC 2012: BPV]
- ✓ (v)-PAKE, LAKE, Secret Handshakes [PKC/Crypto 2013: BBCPV]
- ✓ Oblivious Transfer [AC 2013: ABBCP]

△ Many more Round optimal applications?

Smooth Projective Hash Functions $\triangleq$ *implicit* proofs of knowledge

Various Applications:

- ✓ IND-CCA [CS02]
- ✓ PAKE [GL03]
- ✓ Certification of Public Keys [ACP09]

Privacy-preserving protocols:

- ✓ Blind signatures [TCC 2012: BPV]
- ✓ Oblivious Signature-Based Envelope [TCC 2012: BPV]
- ✓ (v)-PAKE, LAKE, Secret Handshakes [PKC/Crypto 2013: BBCPV]
- ✓ Oblivious Transfer [AC 2013: ABBCP]

△ Many more Round optimal applications?

Groth-Sahai

- Allows to combine efficiently classical building blocks
- Allows several kind of new signatures under standard hypotheses

Smooth Projective Hash Functions

- Can handle more general languages under better hypotheses
- Do not add any extra-rounds in an interactive scenario
- More efficient in the usual cases

Groth-Sahai

- Allows to combine efficiently classical building blocks
- Allows several kind of new signatures under standard hypotheses

Smooth Projective Hash Functions

- Can handle more general languages under better hypotheses
- Do not add any extra-rounds in an interactive scenario
- More efficient in the usual cases

1 General Remarks

2 Building blocks

3 Non-Interactive Proofs of Knowledge

4 Interactive Implicit Proofs

5 Can we do better?
- The problem
- Very high level idea

- We commit to bitstring, bit by bit
- Can we sign a whole message?
- No, we can not extract a scalar
- Can we sign a whole message as a group element?
- Can we do that?

- We commit to bitstring, bit by bit
- Can we sign a whole message?
- No, we can not extract a scalar
- Can we sign a whole message as a group element?
- Can we do that?

- We commit to bitstring, bit by bit
- Can we sign a whole message?
- No, we can not extract a scalar
- Can we sign a whole message as a group element?
- Can we do that?

- We commit to bitstring, bit by bit
- Can we sign a whole message?
- No, we can not extract a scalar
- Can we sign a whole message as a group element?
- Can we do that?

- We commit to bitstring, bit by bit
- Can we sign a whole message?
- No, we can not extract a scalar
- Can we sign a whole message as a group element?
- Can we do that?

Structure Preserving Signature

Original Definition: Signatures composed of group elements, whose public keys are group elements and who signed group elements

Limits

Classical constructions have limits ...
Relies on twisted hypothesis
Have a size linear in $\log p$

Structure Preserving Signature

Original Definition: Signatures composed of group elements, whose public keys are group elements and who signed group elements

Limits

Classical constructions have limits ...
Relies on twisted hypothesis
Have a size linear in $\log p$

Solution

Constant size Structure Preserving Signature (4,1)
Standard hypothesis

But...

It is not randomizable
So need 34,4 elements for the Blind Signatures ...

Thank you..