**Efficient Smooth Projective Hash Functions and Applications**

David Pointcheval

Joint work with Olivier Blazy, Céline Chevalier and Damien Vergnaud

Ecole Normale Supérieure

---

**Motivation**

An authority, or a server, may accept to process a request under some conditions only:

- Certification of public key: if the associated secret key is known
- Transmission of private information: if the receiver owns a credential
- Blind signature on a message: if the user knows the message (for the security proof)

→ Proof of validity/knowledge

Why should the authority learn the final status?

→ Implicit proof of validity/knowledge?

In the **registered key** setting, a user can ask for the certification of a public key $pk$, but only if he knows the associated secret key $sk$:

**With an Interactive Zero-Knowledge Proof of Knowledge**

- the user $U$ sends his public key $pk$;
- $U$ and the authority $A$ run a ZK proof of knowledge of $sk$
- if convinced, $A$ generates and sends the certificate Cert for $pk$

For extracting $sk$ (required in some security proofs), the reduction has to make a rewind (that is not always allowed: *e.g.*, in the UC Framework)

And the authority learns the final status!
Certification of Public Keys: ZK and NIZK Proofs

In the registered key setting, a user can ask for the certification of a public key $pk$, but only if he knows the associated secret key $sk$:

**With an Interactive Zero-Knowledge Proof of Membership**
- the user $U$ sends his public key $pk$, and an encryption $C$ of $sk$;
- $U$ and the authority $A$ run a ZK proof that $C$ contains the secret key $sk$ associated to $pk$
- if convinced, $A$ generates and sends the certificate $Cert$ for $pk$

**With a Non-Interactive Zero-Knowledge Proof of Membership**
- the user $U$ sends his public key $pk$, and an encryption $C$ of $sk$ together with a NIZK proof that $C$ contains the secret key $sk$ associated to $pk$
- if convinced, $A$ generates and sends the certificate $Cert$ for $pk$

Certification of Public Keys: SPHF

In the registered key setting, a user can ask for the certification of a public key $pk$, but only if he knows the associated secret key $sk$:

**With a Smooth Projective Hash Function**
The user $U$ and the authority $A$ use a smooth projective hash system for $L$: $pk$ and $C = \mathcal{E}_{pk}(sk; r)$ are associated to the same $sk$
- the user $U$ sends his public key $pk$, and an encryption $C$ of $sk$;
- $A$ generates the certificate $Cert$ for $pk$, and sends it, masked by $Hash = Hash(hk; (pk, C))$
- $U$ computes $Hash = ProjHash(hp; (pk, C), r))$, and gets $Cert$

Implicit proof of knowledge of $sk$ → the authority does not learn the final status!

Smooth Projective Hash Functions

**Definition**
[Craver, Shoup, 2002] [Gennaro, Lindell, 2003]
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 $hk$: $H(x) = Hash_L(hk; x)$;
- or a public projected key $hp$: $H(x) = ProjHash_L(hp; x, w)$

While the former works for all points in the domain $X$, the latter works for $x \in L$ only, and requires a witness $w$ to this fact.

Public mapping $hk \mapsto hp = ProjKG_L(hk, x)$

For any $x \in X$, $H(x) = Hash_L(hk; x)$
For any $x \in L$, $H(x) = ProjHash_L(hp; x, w)$ $w$ witness that $x \in L$

**Smoothness**
For any $x \not\in L$, $H(x)$ 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 $X$:

**Hard-Partitioned Subset**
$L$ is a hard-partitioned subset of $X$ if it is computationally hard to distinguish a random element in $L$ from a random element in $X \setminus L$
**Introduction**

**Cryptographic Tools**

**More Languages**

**Blind Signatures**

**OSBE**

**LAKE**

---

**Examples**

**DH Language**

Lg,h = {((u, v)) where (g, h, u, v) is DH tuple:

there exists r such that u = gr and v = hr

→ Public-key Encryption with IND-CCA Security

**Algorithms**

- HashKG() = hk = (γ1, γ3) √ Ẑp
- ProjKG(hk) = hp = gy1 cy3
- Hash(hk, (u, v)) = uy1 vy3 = hp = ProjHash(hk, (u, v); r)

---

**Examples (Con’d)**

**Commitment/Encryption**

Lpk,m = {c} where c is an encryption of m under pk:

there exists r such that c = Es(pk)(m; r)

→ Password-Authenticated Key Exchange in the Standard Model

**Labeled Encryption**

Lpk(ℓ, m) = {c} where c is an encryption of m under pk, with label ℓ

→ PAKE in the UC Framework (passive corruptions)

**Extractable/Equivocable Commitment**

Lpk,m = {c} where c is an equivocable/extractable commitment of m

→ PAKE in the UC Framework with Adaptive Corruptions

---

**Assumptions: CDH and DLin**

- A cyclic group of prime order p (with or without bilinear map).

**Definition (The Computational Diffie-Hellman problem (CDH))**

For any generator g $\in G$, and any scalars a, b $\in Z_p^*$,

given (g, ga, gb), compute gab.

Decision variant easy if a bilinear map is available.

**Definition (Decision Linear Problem (DLin))**

For any generator g $\in G$, and any scalars a, b, x, y, c $\in Z_p^*$,

given (g, gx, gy, gxa, gvb, gc), decide whether c = a + b or not.

Equivalently, given a reference triple (u = gx, v = gy, g)

and a new triple (U = ua = gxa, V = vb = gvb, T = gc),

decide whether T = ga+b or not (that is c = a + b).

(U, V, T) is (or not) a linear tuple w.r.t. (u, v, g)

---

**General Tools: Signature**

**Definition (Signature Scheme)**

S = (Setup, KeyGen, Sign, Verif):

- Setup(1k) → global parameters param
- KeyGen(param) → pair of keys (sk, vk)
- Sign(sk, m; s) → signature σ, using the random coins s
- Verif(vk, m, σ) → validity of σ

**Definition (Security: EF-CMA)**

An adversary should not be able to generate a new valid

message-signature pair for a new message (Existential Forgery)
even when having access to any signature of its choice

(Chosen-Message Attack).
**Signature: Waters**

\[ G = \langle g \rangle = \langle h \rangle \text{ group of order } p, \text{ and a bilinear map } e : G \times G \to G_T \]

**Waters Signature**

For a \( k \)-bit message \( M = (M_i) \), we define \( \mathcal{F}(M) = u_0 \prod_{i=1}^{k} u_i^{M_i} \)

- Keys: \( vk = Y = g^x, \ sk = X = h^x, \text{ for } x \xleftarrow{\$} \mathbb{Z}_p \)
- \( Sign(sk = X, M; s), \text{ for } M \in \{0, 1\}^k \text{ and } s \xleftarrow{\$} \mathbb{Z}_p \)

\[ \sigma = (\sigma_1 = X \cdot \mathcal{F}(M)^s, \sigma_2 = g^{-s}) \]

- \( Verif(vk = X, M, \sigma = (\sigma_1, \sigma_2)) \) checks whether

\[ e(g, \sigma_1) \cdot e(\mathcal{F}(M), \sigma_2) = e(Y, h) \]

**Security**

Waters signature reaches EF-CMA under the CDH assumption

---

**General Tools: Encryption**

**Definition (Encryption Scheme)**

\[ \mathcal{E} = (\text{Setup}, \text{KeyGen}, \text{Encrypt}, \text{Decrypt}) : \]

- \( \text{Setup}(1^k) \) \( \rightarrow \) global parameters \( \text{param} \)
- \( \text{KeyGen}(\text{param}) \) \( \rightarrow \) pair of keys \( (pk, dk) \)
- \( \text{Encrypt}(pk, m; r) \) \( \rightarrow \) ciphertext \( c \), using the random coins \( r \)
- \( \text{Decrypt}(dk, c) \) \( \rightarrow \) plaintext, or \( \bot \) if the ciphertext is invalid

**Definition (Security: IND-CPA)**

An adversary should not be able to distinguish the encryption of \( m_0 \) from the encryption of \( m_1 \) \( (\text{Indistinguishability}) \) whereas it can encrypt any message of its choice \( (\text{Chosen-Plaintext Attack}). \)

---

**Encryption: Linear**

\[ G = \langle g \rangle \text{ group of order } p \]

**Linear Encryption**

- Keys: \( dk = (x_1, x_2) \xleftarrow{\$} \mathbb{Z}_p^2, \ pk = (X_1 = g^{x_1}, X_2 = g^{x_2}) \)
- \( Encrypt(pk = (X_1, X_2), M; (r_1, r_2)), \text{ for } M \in G \text{ and } (r_1, r_2) \xleftarrow{\$} \mathbb{Z}_p^2 \)

\[ C = (C_1 = X_1^{r_1}, C_2 = X_2^{r_2}, C_3 = g^{r_1+2r_2} \cdot M) \]
- \( Decrypt(dk = (x_1, x_2), C = (C_1, C_2, C_3)) \rightarrow M = C_3 / C_1^{1/x_1} C_2^{1/x_2} \)

**Security**

Linear encryption reaches IND-CPA under the DLin assumption

---

**Encryption: Linear Cramer-Shoup**

\[ G \text{ group of order } p, \text{ with three independent generators } g_1, g_2, g_3 \in G \]

**Linear Cramer-Shoup Encryption**

- Keys: \( dk = (x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3) \xleftarrow{\$} \mathbb{Z}_p^9 \)

\[ pk = \begin{pmatrix} g_1, c_1 = g_1^{x_1} g_3^{x_2}, c_2 = g_2^{x_1} g_3^{x_2}, c_3 = g_3^{x_1} g_2^{x_2} \cr g_2, d_1 = g_1^{y_1} g_3^{y_2}, d_2 = g_2^{y_1} g_3^{y_2}, \mathcal{H} \cr g_3, h_1 = g_1^{z_1} g_3^{z_2}, h_2 = g_2^{z_1} g_3^{z_2} \end{pmatrix} \]
- \( Encrypt(pk = (g_1, g_2, g_3, c_1, c_2, d_1, d_2, h_1, h_2, \mathcal{H}), m; (r, s)), \text{ for } M \in G \):

\[ C = (\tilde{u} = (u_1 = g_1^{x_1}, u_2 = g_2^{x_2}, u_3 = g_3^{x_3+s}), e = M \cdot h_1^{x_1} h_2^{x_2}, v = v_1^{x_1} v_2^{x_2}) \]

where \( v_1 = c_1^{d_1}, v_2 = c_2^{d_2}, \text{ and } \xi = \mathcal{H}(\tilde{u}, e) \)
- \( Decrypt(dk = (x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3), C = (\tilde{u}, e, v)) \)

one checks \( v = u_1^{x_1} u_2^{x_2} u_3^{x_3} \rightarrow M = e / u_1^{z_1} u_2^{z_2} u_3^{z_3} \)
**Definition (Security: IND-CCA)** [Rackoff, Simon, 1991]

An adversary should not be able to distinguish the encryption of $m_0$ from the encryption of $m_1$ (Indistinguishability) whereas it can encrypt any message of its choice, and ask any decryption of its choice (Chosen-Ciphertext Attack).

**Security: Non-Malleability** [Dolev,Dwork, Naor, 1991]

IND-CCA implies Non-Malleability [Bellare, Desai, Pointcheval, Rogaway, 1998]

**Security of the Linear Cramer-Shoup** [Shacham, 2007]

Linear Cramer-Shoup encryption reaches IND-CCA under the $DLin$ assumption

**Conjunctions and Disjunctions**

**Notations** [Abdalla, Chevalier, Pointcheval, 2009]

We assume that $G$ possesses a group structure, and we denote by $\oplus$ the commutative law of the group (and by $\ominus$ the opposite operation)

We assume to be given two smooth hash systems $SHS_1$ and $SHS_2$, on the sets $G_1$ and $G_2$ (included in $G$) corresponding to the languages $L_1$ and $L_2$ respectively:

$$SHS_i = \{HashKG_i, ProjKG_i, Hash_i, ProjHash_i\}$$

Let $c \in X$, and $r_1$ and $r_2$ two random elements:

$$hk_1 = HashKG_1(r_1) \quad hk_2 = HashKG_2(r_2)$$

$$hp_1 = ProjKG_1(hk_1, c) \quad hp_2 = ProjKG_2(hk_2, c)$$

**Conjunction of Languages**

A hash system for the language $L = L_1 \cap L_2$ is defined as follows, if $c \in L_1 \cap L_2$ and $w_i$ is a witness that $c \in L_i$, for $i = 1, 2$:

$$HashKG_L(r = r_1 \parallel r_2) = hk = (hk_1, hk_2)$$

$$ProjKG_L(hk, c) = hp = (hp_1, hp_2)$$

$$Hash_L(hk, c) = Hash_1(hk_1, c) \oplus Hash_2(hk_2, c)$$

$$ProjHash_L(hp, c; (w_1, w_2)) = ProjHash_1(hp_1, c; w_1) \oplus ProjHash_2(hp_2, c; w_2)$$

- if $c$ is not in one of the languages, then the corresponding hash value is perfectly random: smoothness
- without one of the witnesses, then the corresponding hash value is computationally unpredictable: pseudo-randomness
Disjunction of Languages

A hash system for the language $L = L_1 \cup L_2$ is defined as follows, if $c \in L_1 \cup L_2$ and $w$ is a witness that $c \in L_i$ for $i \in \{1, 2\}$:

$$\text{HashKG}_L(r = r_1 \parallel r_2) = \text{hk} = (\text{hk}_1, \text{hk}_2)$$
$$\text{ProjKG}_L(hk, c) = hp = (hp_1, hp_2, hp_{\Delta})$$
where $hp_{\Delta} = \text{Hash}_1(hk_1, c) \oplus \text{Hash}_2(hk_2, c)$
$$\text{Hash}_L(hk, c) = \text{Hash}_1(hk_1, c)$$
$$\text{ProjHash}_L(hp, c; w) = \text{ProjHash}(hp_1, c; w) \text{ if } c \in L_1$$
$$\text{or } hp_{\Delta} \oplus \text{ProjHash}_2(hp_2, c; w) \text{ if } c \in L_2$$

$hp_{\Delta}$ helps to compute the missing hash value, if and only if at least one can be computed.

Pairing Product Equations

A hash system for the language $L = L_1 \cup L_2$ is defined as follows, if $c \in L_1 \cup L_2$ and $w$ is a witness that $c \in L_i$ for $i \in \{1, 2\}$:

$$\text{HashKG}_L(r = r_1 \parallel r_2) = \text{hk} = (\text{hk}_1, \text{hk}_2)$$
$$\text{ProjKG}_L(hk, c) = hp = (hp_1, hp_2, hp_{\Delta})$$
where $hp_{\Delta} = \text{Hash}_1(hk_1, c) \oplus \text{Hash}_2(hk_2, c)$
$$\text{Hash}_L(hk, c) = \text{Hash}_1(hk_1, c)$$
$$\text{ProjHash}_L(hp, c; w) = \text{ProjHash}(hp_1, c; w) \text{ if } c \in L_1$$
$$\text{or } hp_{\Delta} \oplus \text{ProjHash}_2(hp_2, c; w) \text{ if } c \in L_2$$

$hp_{\Delta}$ helps to compute the missing hash value, if and only if at least one can be computed.

Commissions

$\tilde{c}_i = (u_{i1}^{v_i}, u_{i2}^{s_i}, g_1^{s_i + s_i} \cdot X_i)$ for $i = 1, \ldots, m$
$\tilde{C}_i = (U_{i1}^{v_i}, U_{i2}^{s_i}, G_i^{s_i + s_i} \cdot Z_i)$ for $i = m + 1, \ldots, n$

The $\tilde{c}_i$'s can be transposed into $G_T$, for $i = 1, \ldots, m$:
$$\tilde{c}_i = (u_{i1}^{v_i}, U_{i2}^{s_i}, G_i^{s_i + s_i} \cdot Z_i)$$

where $U_{i1} = e(u_1, A_i)$, $U_{i2} = e(u_2, A_i)$, $G_i = e(g, A_i)$,
but also, $Z_i = e(X_i, A_i)$, for $i = 1, \ldots, m$

We also denote $U_{i1} = U_1$, $U_{i2} = U_2$, $G_i = G$, for $i = m + 1, \ldots, n$

Smooth Projective Hash Function

$$(\lambda, (\eta_i, \theta_i)_{i=1,\ldots,n}) \leftarrow \mathbb{Z}_p^{2n+1}, \text{ one sets } \text{hk}_i = (\eta_i, \theta_i, \lambda)$$
and $hp_i = (u_1^{\lambda g\xi}, u_2^{\lambda g\xi}) \in G_T$
where $\xi_i = 1$ for $i = 1, \ldots, m$.

The associated projection keys in $G_T$ are
$$\text{HP}_i = (e(hp_{i1}, A_i), e(hp_{i2}, A_i))$$, for $i = 1, \ldots, n$,
where $A_i = g$ for $i = m + 1, \ldots, n$.

The hash value is
$$H = \left( \prod_{i=1}^{n} C_{i1}^{\eta_i} \cdot C_{i2}^{\theta_i} \cdot C_{i3}^{\xi_i} \right) \times B^{-\lambda}$$
$$= \left( \prod_{i=1}^{n} \text{HP}_{i1}^{\eta_i} \cdot \text{HP}_{i2}^{\theta_i} \right) \times \left( \prod_{i=1}^{m} e(X_i, A_i) \prod_{i=m+1}^{n} Z_i^{\xi_i} / B \right)^{\lambda}$$

Equality indeed holds if and only if the equation is satisfied.
The easiest way for blind signatures, is to blind the message:

- The user computes a blind version of the hash value:
  \[ M = H(m) \text{ and } M' = M \cdot r^e \mod n \]

- The signer signs \( M' \) into \( \sigma' = M'^d \mod n \)

- The user unblinds the signature: \( \sigma = \sigma' / r \mod n \)

Indeed,

\[ \sigma = \frac{\sigma'}{r} = \frac{M'^d}{r} = (M \cdot r^e)^d \cdot r^d = M^d \cdot r / r = M^d \mod n \]

→ Proven under the One-More RSA

[Bellare, Namprempre, Pointcheval, Semanko, 2001]
Blind Signatures

Such a primitive can be used for a Waters Blind Signature, by encrypting $F(M)$:

- Unforgeability: one-more forgery would imply a forgery against the signature scheme (CDH assumption)
- Blindness: a distinguisher would break indistinguishability of the encryption scheme (DLin assumption)

Efficiency
One obtains a plain Waters Signature

Limitation
A proof of knowledge of $M$ in $C = Epk(F(M))$ has to be sent

In order to get the $\ell$-bit message $M = \{M_i\}$ blindly signed:

With Groth-Sahai NIZKP
- the user $U$ encrypts $M$ into $C_1$, and $F(M)$ into $C_2$;
- $U$ produces a Groth-Sahai NIZK Proof that $C_1$ and $C_2$ contain the same $M$ (bit-by-bit proof)
- if convinced, $A$ generates a signature on $C_2$
- granted the commutativity, $U$ decrypts it into a Waters signature of $M$, and eventually re-randomizes the signature

Such a protocol requires $8\ell + 12$ group elements in total only!

Oblivious Transfers

In order to get the $\ell$-bit message $M = \{M_i\}$ blindly signed:

With SPHF
The user $U$ and the authority $A$ use a smooth projective hash system for $L$: $C_1 = E_{pk_1}(M; r)$ and $C_2 = E_{pk_2}(F(M); s)$ contain the same $M$
- $U$ sends encryptions of $M$, into $C_1$, and $F(M)$, into $C_2$;
- $A$ generates
  - a signature $\sigma$ on $C_2$,
  - masks it using $\text{Hash} = \text{Hash}(hk; (C_1, C_2))$
- $U$ computes $\text{Hash} = \text{ProjHash}(hp; (C_1, C_2), (r, s))$, and gets $\sigma$.

Granted the commutativity, $U$ decrypts it into a Waters signature of $M$, and eventually re-randomizes it

Such a protocol requires $8\ell + 12$ group elements in total only!
**Oblivious Signature-Based Envelope**  
[Li, Du, Boneh, 2003]

A sender $S$ wants to send a message $M$ to $U$ such that
- $U$ gets $M$ if and only if it owns a signature $\sigma$ on a message $m$ valid under $vk$
- $S$ does not learn whereas $U$ gets the message $M$ or not

**Security Notions**
- **Oblivious**: $S$ does not know whether $U$ owns a valid signature (and thus gets the message)
- **Semantic Security**: $U$ does not learn any information about $M$ if he does not own a valid signature

**Correctness**: if $U$ owns a valid signature, he learns $M$

---

**Security Properties**
- **Oblivious (even w.r.t. the Authority)**: IND-CPA of the encryption scheme (Hard-partitioned Subset of the SPHF);
- **Semantic Security**: Smoothness of the SPHF
- **Semantic Security w.r.t. the Authority**: Pseudo-randomness of the SPHF

Semantic Security w.r.t. the Authority requires one interaction $\rightarrow$ round-optimal

Standard model with Waters Signature + Linear Encryption $\rightarrow$ CDH and DLin assumptions

---

**A New OSBE**  
[Blazy, Pointcheval, Vergnaud, 2012]

A stronger security model

S wants to send a message $M$ to $U$, if $U$ owns/uses a valid signature.

**Security Notions**
- **Oblivious w.r.t. the authority**: the authority does not know whether $U$ uses a valid signature (and thus gets the message);
- **Semantic Security**: $U$ cannot distinguish multiple interactions with $S$ sending $M_0$ from multiple interactions with $S$ sending $M_1$ if he does not own/use a valid signature;
- **Semantic Security w.r.t. the Authority**: after the interaction, the authority does not learn any information about $M$.

---

**Our Scheme**

The user $U$ and the sender $S$ use a smooth projective hash system for $L$: $C = E_{pk} (\sigma; r)$ contains a valid signature $\sigma$ of $m$ under $vk$
- the user $U$ sends an encryption $C$ of $\sigma$;
- $S$ generates a $hk$ and the associated $hp$, computes $\text{Hash} = \text{Hash}(hk; C)$, and sends $hp$ together with $c = M \oplus \text{Hash}$;
- $U$ computes $\text{Hash} = \text{ProjHash}(hp; C, r)$, and gets $M$.
General Construction

- The user $U$ sends a commitment $C$ of a word $w$
- $S$ generates a $hk$ and the associated $hp$, computes $\text{Hash} = \text{Hash}(hk; C)$, and sends $hp$ together with $c = M \oplus \text{Hash}$;
- $U$ computes $\text{Hash} = \text{ProjHash}(hp; C, r)$, and gets $M$.

$U$ gets $M$ iff $w$ is in the appropriate language:
- a signature on a public message: OSBE
- a signature on a private message: Anonymous Credential
- a private message (low entropy): Password

Password-based Authenticated Key Exchange

**GL - Generic Approach**

- Alice
  - $C_1 = \text{Commit}(pw; r_1)$
  - $C_2 = \text{Commit}(pw; r_2)$
- Bob
  - $h_{k_1}, h_{p_1}$ on $C_1$
  - $h_{k_2}, h_{p_2}$ on $C_2$

$\text{ProjHash}(h_{p_1}; C_1, r_1) = H_1 = \text{Hash}(h_{k_1}; C_1)$
$\text{Hash}(h_{k_2}; C_2) = H_2 = \text{ProjHash}(h_{p_2}; C_2, r_2)$

$K = H_1 \cdot H_2$

The language is: valid commitments of $pw$

Languages

- Password: PAKE secure under $DLin$
- Waters Signature: Secret Handshake, Credentials secure under $DLin + CDH$

Any Linear Pairing Product Equation Systems in both $G$ and $G_T$
Smooth Projective Hash Functions can be used as implicit proofs of knowledge or membership

Various Applications
- IND-CCA  \[\text{[Cramer, Shoup, 2002]}\]
- PAKE  \[\text{[Gennaro, Lindell, 2003]}\]
- Certification of Public Keys  \[\text{[Abdalla, Chevalier, Pointcheval, 2009]}\]

Privacy-preserving protocols
- Blind signatures  \[\text{[Blazy, Pointcheval, Vergnaud, 2012]}\]
- Oblivious Signature-Based Envelope
  \[\rightarrow \text{ Round optimal!}\]

More general: Language-based Authenticated Key Exchange