Efficient Smooth Projective Hash Functions and Applications

David Pointcheval

Joint work with Olivier Blazy, Céline Chevalier and Damien Vergnaud

Ecole Normale Supérieure

Motivation

Conditional Actions

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?

Certification of Public Keys: ZKPoK

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 $C = E_{pk}(sk; r)$ 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 $\text{Hash} = \text{Hash}(hk; (pk, C))$;
- $U$ computes $\text{Hash} = \text{ProjHash}(hp; (pk, C, r))$, and gets Cert.

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

**Smooth Projective Hash Functions**

**Definition**

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 $hk$: $H(x) = \text{Hash}_L(hk; x)$;
  - or a public projected key $hp$: $H(x) = \text{ProjHash}_L(hp; x, w)$

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 $hk \mapsto hp = \text{ProjKG}_L(hk, x)$

**Properties**

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

**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 $X$:

$L$ is a hard-partitioned subset of $X$ if it is computationally hard to distinguish a random element in $L$ from a random element in $X \setminus L$
# Examples

**DH Language**

\[ L_{g,h} = \{ (u,v) \} \text{ where } (g,h,u,v) \text{ is DH tuple:} \]
\[ \text{there exists } r \text{ such that } u = g^r \text{ and } v = h^r \]

→ Public-key Encryption with IND-CCA Security

**Algorithms**

- HashKG = \( h_k = (\gamma_1, \gamma_3) \rightarrow \mathbb{Z}_p^2 \)

- ProjKG(hk) = \( h_p = g^\gamma_1 h_3 \)

\[ \text{Hash}(hk,(u,v)) = u^\gamma_1 v_3 = h_p = \text{ProjHash}(hp,(u,v);r) \]

# Assumptions: CDH and DLin

\( G \) a cyclic group of prime order \( p \) (with or without bilinear map).

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

\[ \text{For any generator } g \leftarrow G, \text{ and any scalars } a, b \leftarrow \mathbb{Z}_p^* \]
\[ \text{given } (g, g^a, g^b), \text{ compute } g^{ab}. \]

Decision variant easy if a bilinear map is available.

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

\[ \text{For any generator } g \leftarrow G, \text{ and any scalars } a, b, x, y, c \leftarrow \mathbb{Z}_p^* \]
\[ \text{given } (g, g^x, g^y, g^{xa}, g^{yb}, g^c), \text{ decide whether } c = a + b. \]

Equivalently, given a reference triple \( (u = g^x, v = g^y, g) \)
and a new triple \( (U = u^a = g^{xa}, V = v^b = g^{yb}, T = g^c) \),
decide whether \( T = g^{a+b} \) or not (that is \( c = a + b \)).

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

# Applications

**Examples**

**Examples (Con’d)**

**Commitment/Encryption**

\[ L_{pk,m} = \{ c \} \text{ where } c \text{ is an encryption of } m \text{ under } pk: \]
\[ \text{there exists } r \text{ such that } c = \mathcal{E}_{pk}(m; r) \]

→ Password-Authenticated Key Exchange in the Standard Model

**Labeled Encryption**

\[ L_{pk,(\ell,m)} = \{ c \} \text{ where } c \text{ is an encryption of } m \text{ under } pk, \text{ with label } \ell \]

→ PAKE in the UC Framework (passive corruptions)

**Extractable/Equivocable Commitment**

\[ L_{pk,m} = \{ c \} \text{ where } c \text{ is an equivalent/extractable commitment of } m \]

→ PAKE in the UC Framework with Adaptive Corruptions

**General Tools: Signature**

**Definition (Signature Scheme)**

\[ \mathcal{S} = (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{Verif}) : \]
\[ \text{Setup}(1^k) \rightarrow \text{global parameters } \text{param} \]
\[ \text{KeyGen}(\text{param}) \rightarrow \text{pair of keys } (sk, vk) \]
\[ \text{Sign}(sk, m, s) \rightarrow \text{signature } \sigma, \text{ using the random coins } s \]
\[ \text{Verif}(vk, m, \sigma) \rightarrow \text{validity of } \sigma \]

**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**

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

**Waters Signature**

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

- **Keys**: \( vk = (Y, X), sk = (h, s) \), for \( X \leftarrow Z_p \)
- **Sign**: \( sk = (X, M; s), M \in \{0, 1\}^k \) and \( s \leftarrow \mathbb{Z}_p \)
  \[ \sigma = (\sigma_1 = X \cdot F(M)^s, \sigma_2 = g^{-s}) \]
- **Verif**: \( \psi = (X, M, \sigma) = (\sigma_1, \sigma_2) \) checks whether \( e(g, \sigma_1) \cdot e(F(M), \sigma_2) = e(Y, h) \)

**Security**

Waters signature reaches EF-CMA under the CDH assumption

---

**Encryption: Linear**

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

**Linear Encryption**

- **Keys**: \( dk = (x_1, x_2) \leftarrow Z_p, pk = (X_1 = g^{x_1}, X_2 = g^{x_2}) \)
- **Encrypt**: \( (X_1, X_2, M; (r_1, r_2)), M \in G \) and \( (r_1, r_2) \leftarrow \mathbb{Z}_p^2 \)
  \[ C = (C_1 = X_1^{r_1}, C_2 = X_2^{r_1}, C_3 = g^{r_1+r_2} \cdot M) \]
- **Decrypt**: \( (x_1, x_2), (C_1, C_2, C_3)) \rightarrow M = C_3 / C_1^{x_1} C_2^{1/x_2} \)

**Security**

Linear encryption reaches IND-CPA under the DLin assumption

---

**Definition (Encryption Scheme)**

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

- **Setup**\( (1^k) \rightarrow \) global parameters \( \text{param} \)
- **KeyGen**\( (\text{param}) \rightarrow \) pair of keys \( (pk, dk) \)
- **Encrypt**\( (pk, m; r) \rightarrow \) ciphertext \( c \), using the random coins \( r \)
- **Decrypt**\( (dk, c) \rightarrow \) plaintext, or \( \perp \) 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 \( m_1 \) (Indistinguishability) whereas it can encrypt any message of its choice (Chosen-Plaintext Attack).

---

**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**: \( dk = (x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3) \leftarrow \mathbb{Z}_p^9 \)
  \[ pk = \left( \begin{array}{c} g_1 \, c_1 = g_1^{x_1} g_3^{x_3}, c_2 = g_2^{x_2} g_3^{x_3} \\ g_2 \, d_1 = g_1^{y_1} g_3^{y_3}, d_2 = g_2^{y_2} g_3^{y_3}, h_1 = g_1^{z_1} g_3^{z_3}, h_2 = g_2^{z_2} g_3^{z_3} \end{array} \right) \]
- **Encrypt**\( (pk, (g_1, g_2, g_3, c_1, c_2, d_1, d_2, h_1, h_2, m; (r, s)), for M \in G \)
  \[ C = (u = (u_1 = g_1^{r_1} u_2 = g_2^{s_1} u_3 = g_3^{s_1+s}) \text{, } e = M \cdot h_1^{r_1} h_2^{s_1} v = v_1^{s_1} v_2^{s_2}) \]
  where \( v_1 = c_1 d_1^e, v_2 = c_2 d_2^e, \text{ and } e = \mathcal{H}(u, e) \)
- **Decrypt**\( (dk, (x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3), (u, e, v)) \)
  one checks \( v = u_1^{x_1+\xi y_1} u_2^{x_2+\xi y_2} u_3^{x_3+\xi y_3} \rightarrow M = e/u_1^{z_1} u_2^{z_2} u_3^{z_3} \)
**Encryption: CCA Security**

**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

---

**Groth-Sahai Methodology**

**Groth-Sahai Proofs** [Groth, Sahai, 2008]

For any pairing product equation of the form:

\[
\prod e(A_i, X_i)^{\alpha_i} \prod e(X_i, X_j)^{\gamma_{ij}} = e(A, B),
\]

where the \( A, B, A_i \in G \) are constant group elements, \( \alpha_i \in \mathbb{Z}_p \) and \( \gamma_{ij} \in \mathbb{Z}_p \) are constant scalars, and \( X_i \) are unknowns

- either group elements in \( G \),
- or of the form \( g^{\gamma_i} \),

one can make a proof of knowledge of values for the \( X_i \)'s or \( x_i \)'s so that the equation is satisfied:

- one first commits these secret values using random coins,
- and then provides proofs, that are group elements, using the above random coins,

\[ \rightarrow \ \text{Under the } DLin \text{ assumption: Efficient NIZK} \]

---

**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:

\[
SHS_i = \{ \text{HashKG}_i, \text{ProjKG}_i, \text{Hash}_i, \text{ProjHash}_i \}
\]

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

\[
\begin{align*}
hk_1 &= \text{HashKG}_1(r_1) & hk_2 &= \text{HashKG}_2(r_2) \\
hp_1 &= \text{ProjKG}_1(hk_1, c) & hp_2 &= \text{ProjKG}_2(hk_2, c)
\end{align*}
\]

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 \( i = 1, 2 \):

\[
\begin{align*}
\text{HashKG}_L(r) &= r_1 \parallel r_2 = hk = (hk_1, hk_2) \\
\text{ProjKG}_L(hk, c) &= hp = (hp_1, hp_2) \\
\text{Hash}_L(hk, c) &= \text{Hash}_1(hk_1, c) \oplus \text{Hash}_2(hk_2, c) \\
\text{ProjHash}_L(hp, c; (w_1, w_2)) &= \text{ProjHash}_1(hp_1, c; w_1) \oplus \text{ProjHash}_2(hp_2, c; w_2)
\end{align*}
\]

- 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
Blind Signatures

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 \( i \in \{1, 2\} \):

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

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

Pairing Product Equations

Smooth Projective Hash Function

\((\lambda, (\eta_i, \theta_i)_{i=1,...,n}) \leftarrow \mathbb{Z}_p^{2n+1}, \) one sets \( hk_i = (\eta_i, \theta_i, \lambda) \)
and \( hp_i = (u_i^{\eta_i} g_i^{\theta_i}, u_i^{\theta_i} g_i^{\lambda}) \in \mathbb{G}^2 \)
where \( \lambda_i = 1 \) for \( i = 1, \ldots, m \).

The associated projection keys in \( \mathbb{G}_T \) are
\[
\text{HP}_i = (e(hp_{i,1}, A_i), e(hp_{i,2}, A_i)), \quad \text{for } i = 1, \ldots, n,
\]
where \( A_i = g \) for \( i = m + 1, \ldots, n \).

The hash value is
\[
H = \left( \prod_{i=1}^{n} C_{i,1}^{\eta_i} \cdot C_{i,2}^{\theta_i} \cdot C_{i,3}^{\lambda} \right) \times B^{-\lambda} \\
= \left( \prod_{i=1}^{n} \text{HP}_{i,1}^{\eta_i} \text{HP}_{i,2}^{\theta_i} \right) \times \left( \prod_{i=1}^{m} e(X_i, A_i) \prod_{i=m+1}^{n} Z_i^{\lambda_i} / B \right)^\lambda
\]

Equality indeed holds if and only if the equation is satisfied.
Blind Signatures

We have $X_i$ committed in $G_i$ in $C_i$, for $i = 1, \ldots, m$ and $Z_i$ committed in $G_T$, in $\tilde{C}_i$, for $i = m + 1, \ldots, n$. We want to show they simultaneously satisfy

$$\left( \prod_{i \in A_k} e(X_i, A_{k,i}) \right) \cdot \left( \prod_{i \in B_k} Z_i^{\zeta_{k,i}} \right) = B_k, \text{ for } k = 1, \ldots, t$$

where $A_{k,i} \in G_i$, $B_k \in G_T$, and $\zeta_{k,i} \in \mathbb{Z}_p$ are public, as well as $A_k \subseteq \{1, \ldots, m\}$ and $B_k \subseteq \{m + 1, \ldots, n\}$

This is a conjunction of languages

$\rightarrow$ Similar Hash Proofs on Linear Cramer-Shoup Commitments

Randomizable Commutative Signature/Encryption

Blind Signatures

Randomizable Commutative Signature/Encryption

[Blazy, Fuchsbauer, Pointcheval, Vergnau, 2011]

- The user "blinds" $M$ into $C$, under random coins $r$
- The signer signs $C$ into $\sigma(C)$, under random coins $s$
- The user "unblinds" the signature $\sigma(M)$, granted the coins $r$

Weakness

The signer can recognize his signature: the random coins $s$ in $\sigma(M)$

$\rightarrow$ Randomizable Signature

Security

- Encryption hides $M$ (blinding of the message)
- Re-randomization hides $\sigma(M)$ (blinding of the signature)

Randomizable Commutative Signature/Encryption

[Blazy, Fuchsbauer, Pointcheval, Vergnau, 2011]

The easiest way for blind signatures, is to blind the message:

To get an RSA signature on $m$ under public key $(n, e)$,

- The user computes a blind version of the hash value: $M = H(m)$ 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,

$$\sigma = \sigma' / r = M'^d / r = (M \cdot r^e)^d / r = M^d \cdot r / r = M^d \mod n$$

$\rightarrow$ Proven under the One-More RSA

[Blazzeo, Namprempre, Pointcheval, Semanko, 2001]
Our Constructions

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 \( C = \varepsilon_{pk}(F(M)) \) has to be sent

Blind Signature

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

9\( \ell \) + 24 group elements have to be sent:
  \( \rightarrow \) It was the most efficient blind signature up to 2011
Why NIZK, since there are already two flows?

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 \( L \): \( C_1 = \varepsilon_{pk_1}(M; r) \) and \( C_2 = \varepsilon_{pk_2}(F(M); s) \) 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 Hash = Hash(hk; (\( C_1, C_2 \))
- \( U \) computes Hash = 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

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

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

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

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

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
General More Languages

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

Language-based Authenticated Key Exchange

**Definition**

- Alice owns a word $w_1$ is a language $L_1(Pub_1, Priv_1)$;
- Bob owns a word $w_2$ is a language $L_2(Pub_2, Priv_2)$;
- If Alice and Bob agree on the languages, and actually own valid words (implicit authentication), they will agree on a common session key (semantic security)

- $Pub = \emptyset$, $Priv = pw$ and $L(Pub, Priv) = \{Priv\}$: PAKE
- $Pub = M$, $Priv = vk$, $L(Pub, Priv) = \{\sigma, \text{Verif}(Priv, Pub, \sigma) = 1\}$: Secret Handshake
- $Pub = \emptyset$, $Priv = (vk, M)$, $L(Pub, Priv) = \{\sigma, \text{Verif}(Priv, \sigma) = 1\}$: CAKE – Credential-based AKE

Our Construction

- With a Linear Cramer-Shoup UC commitment
- Using the GL approach
  → UC Secure LAKE

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 $G_T$

---

**Password-based Authenticated Key Exchange**

**GL – Generic Approach** [Gennaro, Lindell, 2003]

Additional tricks are required for the security!

Alice sends $\mathbf{C}_1 = \text{Commit}(pw; r_1)$

Bob sends $\mathbf{C}_2 = \text{Commit}(pw; r_2)$

$C_1 \rightarrow C_2 : \mathbf{C}_2, hp_1$

$\mathbf{C}_2, hp_1$ on $C_1$

$hk_2, hp_2$ on $C_2$

$hp_2 \leftarrow \text{ProjHash}(hp_1; C_1, r_1)$

$\text{Hash}(hk_1; C_1) = H_1$

$\text{Hash}(hk_2; C_2) = H_2 = \text{ProjHash}(hp_2; C_2, r_2)$

$K = H_1 \cdot H_2$

The language is: valid commitments of $pw$
Smooth Projective Hash Functions can be used as implicit proofs of knowledge or membership

Various Applications

- IND-CCA [Cramer, Shoup, 2002]
- PAKE [Gennaro, Lindell, 2003]
- Certification of Public Keys [Abdalla, Chevalier, Pointcheval, 2009]

Privacy-preserving protocols

- Blind signatures [Blazy, Pointcheval, Vergnaud, 2012]
- Oblivious Signature-Based Envelope

→ Round optimal!

More general: Language-based Authenticated Key Exchange