Smooth Projective Hashing for Conditionally Extractable Commitments

David Pointcheval

Joint work with Michel Abdalla and Céline Chevalier
Ecole normale supérieure, CNRS & INRIA

NTT – Tokyo – Japan
April 10th, 2009

Outline

1. Smooth Projective Hash Functions
   - Definitions
   - Conjunctions and Disjunctions
2. Extractable Commitments
   - Properties
   - Conditional Extractability
   - Application: Certification of Public Keys
3. Equivocable and Extractable Commitments
   - Description
   - Analysis
4. Adaptive Security and UC PAKE
   - Universal Composability
   - Previous Schemes
   - Our Scheme

Outline

1. Smooth Projective Hash Functions
   - Definitions
   - Conjunctions and Disjunctions
2. Extractable Commitments
   - Properties
   - Conditional Extractability
   - Application: Certification of Public Keys
3. Equivocable and Extractable Commitments
   - Description
   - Analysis
4. Adaptive Security and UC PAKE
   - Universal Composability
   - Previous Schemes
   - Our Scheme

Smooth Projective Hash Functions [Cramer-Shoup EC '02]

Family of Hash Function H

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

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. There is a public mapping that converts the hashing key hk into the projected key hp: $\mathrm{hp} = \mathrm{ProjKG}_L(\mathrm{hk})$

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 $x \in L$

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:

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 $X \setminus L$

Element-Based Projection

Initial Definition [Cramer-Shoup EC '02]

The projected key hp depends on the hashing key hk only:
$\mathsf{hp} = \mathsf{ProjKG}_L(\mathsf{hk})$

New Definition [Gennaro-Lindell EC '03]

The projected key hp depends on the hashing key hk, and x:
$\mathsf{hp} = \mathsf{ProjKG}_L(\mathsf{hk}, x)$

Applications: Encryption and Commitments

The input x can be a ciphertext or a commitment,
where the indistinguishability for the hard partitioned subset relies

- either on the semantic security of the encryption scheme
- or the hiding property of the commitment scheme

Examples

Commitment [Gennaro-Lindell EC '02]

$L_{\mathsf{pk},m} = \{c\}$ such that c is a commitment of m
using public parameter pk:
there exists r such that $c = \mathsf{com}_{\mathsf{pk}}(m; r)$
where com is the committing algorithm

Labeled Encryption [Canetti-Halevi-Katz-Lindell-MacKenzie EC '05]

$L_{\mathsf{pk},(\ell,m)} = \{c\}$ such that c is an encryption of m
with label ℓ, under the public key pk:
there exists r such that $c = \mathcal{E}^{\ell}_{pk}(m; r)$
where $\mathcal{E}$ is the encryption algorithm

Smooth Projective Hash Functions [Gennaro-Lindell EC '03]

A family of smooth projective hash functions **HASH**(pk),
for a language $L_{\mathsf{pk},aux} \subset X$, onto the set G, based on

- either a labeled encryption scheme with public key pk
- or on a commitment scheme with public parameters pk

consists of four algorithms:
$\mathbf{HASH}(\mathsf{pk}) = (\mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash})$

Key-Generation Algorithms

- Probabilistic hashing key algorithm:
  $\mathsf{hk} \stackrel{\$}{\leftarrow} \mathsf{HashKG}(\mathsf{pk}, aux)$
- Deterministic projection key algorithm
  $\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}; \mathsf{pk}, aux, c)$
  (where c is either a ciphertext or a commitment in X)

Smooth Projective Hash Functions

[Gennaro-Lindell EC '03]

$$\mathbf{HASH}(\mathsf{pk}) = (\mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash})$$

Hashing Algorithms

- The hashing algorithm Hash computes,
  - on $c \in X$
  - using the secret hashing key hk
  - the value $g = \mathsf{Hash}(\mathsf{hk}; \mathsf{pk}, aux, c) \in G$
- The projected hashing algorithm ProjHash computes,
  - on $c \in X$
  - using the projection key hp
  - and a witness w to the fact that $c \in L_{\mathsf{pk},aux}$
  - the value $g = \mathsf{ProjHash}(\mathsf{hp}; \mathsf{pk}, aux, c; w) \in G$

Properties

Correctness

Let $c \in L_{\mathsf{pk},aux}$ and w a witness of this membership.
$\mathsf{hk} \stackrel{\$}{\leftarrow} \mathsf{HashKG}(\mathsf{pk}, aux)$ and $\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}; \mathsf{pk}, aux, c)$ implies

$$\mathsf{Hash}(\mathsf{hk}; \mathsf{pk}, aux, c) = \mathsf{ProjHash}(\mathsf{hp}; \mathsf{pk}, aux, c; w)$$

Smoothness

If $c \notin L_{\mathsf{pk},aux}$, the two distributions are statistically indistinguishable:

$$\{\mathsf{pk}, aux, c, \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}; \mathsf{pk}, aux, c), \quad g = \mathsf{Hash}(\mathsf{hk}; \mathsf{pk}, aux, c)\}$$
$$\{\mathsf{pk}, aux, c, \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}; \mathsf{pk}, aux, c), \quad g \stackrel{\$}{\leftarrow} G\}$$

Properties

Pseudorandomness

If $c \in L_{\mathsf{pk},aux}$, without a witness w of this membership, the two distributions are computationally indistinguishable:

$$\{\mathsf{pk}, aux, c, \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}; \mathsf{pk}, aux, c), \quad g = \mathsf{Hash}(\mathsf{hk}; \mathsf{pk}, aux, c)\}$$
$$\{\mathsf{pk}, aux, c, \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}; \mathsf{pk}, aux, c), \quad g \stackrel{\$}{\leftarrow} G\}$$

This requires $L_{\mathsf{pk},aux}$ to be a hard partitioned subset of X:
the uniform distributions in $L_{\mathsf{pk},aux}$ and in $X \backslash L_{\mathsf{pk},aux}$ are computationally indistinguishable

ElGamal Encryption

[ElGamal - C '84]

$G = \langle g \rangle$, a cyclic group of prime order q.

ElGamal Encryption Schemes

Let $\mathsf{pk} = h = g^x$ (public key), where $\mathsf{sk} = x \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ (private key)

- If $M \in G$, the multiplicative ElGamal encryption is:
  - $\mathsf{EG}^{\times}_{\mathsf{pk}}(M; r) = (u_1 = g^r, e = h^r M)$
  - which can be decrypted by $M = e/u_1^z$.
- If $M \in \mathbb{Z}_q$, the additive ElGamal encryption is:
  - $\mathsf{EG}^{+}_{\mathsf{pk}}(M; r) = (u_1 = g^r, e = h^r g^M)$
    Note that $\mathsf{EG}^{\times}_{\mathsf{pk}}(g^M; r) = \mathsf{EG}^{+}_{\mathsf{pk}}(M; r)$
  - It can thus be decrypted as above, but after an additional discrete logarithm computation: M must be small enough.

IND-CPA security = DDH assumption.

Smooth Projective HF Family for ElGamal

The CRS: $\rho = (G, q, g, \mathsf{pk} = h)$

Language: $L = L_{(\mathbf{EG}^+,\rho),M} = \{C = (u_1, e) = \mathsf{EG}^+_{\mathsf{pk}}(M; r), r \stackrel{\$}{\leftarrow} \mathbb{Z}_q\}$

- L is a hard partitioned subset of $X = G^2$, under the semantic security of the ElGamal encryption scheme (DDH assumption)
- the random r is the witness to L-membership

Algorithms

- $\mathsf{HashKG}((\mathbf{EG}^+, \rho), M) = \mathsf{hk} = (\gamma_1, \gamma_3) \stackrel{\$}{\leftarrow} \mathbb{Z}_q \times \mathbb{Z}_q$
- $\mathsf{Hash}(\mathsf{hk}; (\mathbf{EG}^+, \rho), M, C) = (u_1)^{\gamma_1}(eg^{-M})^{\gamma_3}$
- $\mathsf{ProjKG}(\mathsf{hk}; (\mathbf{EG}^+, \rho), M, C) = \mathsf{hp} = (g)^{\gamma_1}(h)^{\gamma_3}$
- $\mathsf{ProjHash}(\mathsf{hp}; (\mathbf{EG}^+, \rho), M, C; r) = (\mathsf{hp})^r$

Notations

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:

$$\mathsf{SHS}_i = \{\mathsf{HashKG}_i, \mathsf{ProjKG}_i, \mathsf{Hash}_i, \mathsf{ProjHash}_i\}$$

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

$$\begin{aligned}
\mathsf{hk}_1 &= \mathsf{HashKG}_1(\rho, aux, r_1)\\
\mathsf{hk}_2 &= \mathsf{HashKG}_2(\rho, aux, r_2)\\
\mathsf{hp}_1 &= \mathsf{ProjKG}_1(\mathsf{hk}_1; \rho, aux, c)\\
\mathsf{hp}_2 &= \mathsf{ProjKG}_2(\mathsf{hk}_2; \rho, aux, c)
\end{aligned}$$

Conjunction of Languages

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

$$\begin{aligned}
\mathsf{HashKG}_L(\rho, aux, r = r_1 \| r_2) &= \mathsf{hk} = (\mathsf{hk}_1, \mathsf{hk}_2)\\
\mathsf{ProjKG}_L(\mathsf{hk}; \rho, aux, c) &= \mathsf{hp} = (\mathsf{hp}_1, \mathsf{hp}_2)\\
\mathsf{Hash}_L(\mathsf{hk}; \rho, aux, c) &= \mathsf{Hash}_1(\mathsf{hk}_1; \rho, aux, c)\\
&\quad \oplus \mathsf{Hash}_2(\mathsf{hk}_2; \rho, aux, c)\\
\mathsf{ProjHash}_L(\mathsf{hp}; \rho, aux, c; (w_1, w_2)) &= \mathsf{ProjHash}_1(\mathsf{hp}_1; \rho, aux, c; w_1)\\
&\quad \oplus \mathsf{ProjHash}_2(\mathsf{hp}_2; \rho, aux, c; w_2)
\end{aligned}$$

- 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 then defined as follows, if $c \in L_1 \cup L_2$ and w is a witness that $c \in L_i$ for $i \in \{1, 2\}$:

$$\begin{aligned}
\mathsf{HashKG}_L(\rho, aux, r = r_1 \| r_2) &= \mathsf{hk} = (\mathsf{hk}_1, \mathsf{hk}_2)\\
\mathsf{ProjKG}_L(\mathsf{hk}; \rho, aux, c) &= \mathsf{hp} = (\mathsf{hp}_1, \mathsf{hp}_2, \mathsf{hp}_\Delta)\\
\text{where } \mathsf{hp}_\Delta &= \mathsf{Hash}_1(\mathsf{hk}_1; \rho, aux, c)\\
&\quad \oplus \mathsf{Hash}_2(\mathsf{hk}_2; \rho, aux, c)\\
\mathsf{Hash}_L(\mathsf{hk}; \rho, aux, c) &= \mathsf{Hash}_1(\mathsf{hk}_1; \rho, aux, c)\\
\mathsf{ProjHash}_L(\mathsf{hp}; \rho, aux, c; w) &= \mathsf{ProjHash}_1(\mathsf{hp}_1; \rho, aux, c; w) \text{ if } c \in L_1\\
&\quad \text{or } \mathsf{hp}_\Delta \ominus \mathsf{ProjHash}_2(\mathsf{hp}_2; \rho, aux, c; w)\\
&\qquad\qquad \text{if } c \in L_2
\end{aligned}$$

hp_Δ helps to compute the missing hash value, if and only if at least one can be computed

Properties

Contrarily to the original Cramer-Shoup definition,
the value of the projected key formally depends on the word c
But this dependence maybe *invisible*

Uniformity

The projected key may or may not depend on c (and aux),
but its distribution does not

Independence

The projected key does not depend at all on c (and aux)

Outline

1. Smooth Projective Hash Functions
   - Definitions
   - Conjunctions and Disjunctions
2. Extractable Commitments
   - Properties
   - Conditional Extractability
   - Application: Certification of Public Keys
3. Equivocable and Extractable Commitments
   - Description
   - Analysis
4. Adaptive Security and UC PAKE
   - Universal Composability
   - Previous Schemes
   - Our Scheme

Commitments

Definition

A commitment scheme is defined by two algorithms:

- the committing algorithm, $C = \mathrm{com}(x; r)$ with randomness r, on input x, to commit on this input;
- the decommitting algorithm, $(x, D) = \mathrm{decom}(C, x, r)$, where x is the claimed committed value, and D the proof

Properties

The commitment $C = \mathrm{com}(x; r)$

- reveals nothing about the input x: the hiding property
- nobody can open C in two different ways: the binding property

Examples

In both cases, the CRS ρ is $(G, q, g, \mathrm{pk} = h)$,
and $(x, D = r) = \mathrm{decom}(C, x, r)$

ElGamal

- $C = \mathrm{comEG}_{\mathrm{pk}}(x; r) = (u_1, e) = \mathrm{EG}^{+}_{\mathrm{pk}}(x; r)$, with $r \stackrel{\$}{\leftarrow} \mathbb{Z}_q$;
- As any IND-CPA encryption scheme, this commitment is perfectly binding and computationally hiding, (DDH assumption)

Pedersen

- $C = \mathrm{comPed}(x; r) = g^x h^r$, with $r \stackrel{\$}{\leftarrow} \mathbb{Z}_q$;
- This commitment is perfectly hiding and computationally binding, (DL assumption)

Additional Properties

Extractability

A commitment is extractable if there exists an efficient algorithm, called extractor, capable of generating a new CRS (with similar distribution) such that it can extract x from any $C = \mathsf{com}(x, r)$

This is possible for computationally hiding commitments only: with an encryption scheme, the decryption key is the extraction key

Equivocability

A commitment is equivocable if there exists an efficient algorithm, called equivocator, capable of generating a new CRS and a commitment (with similar distributions) such that the commitment can be opened in different ways

This is possible for computationally binding commitments only

Additional Properties

Non-Malleability

A commitment is non-malleable if, for any adversary receiving a commitment C of some unknown value x that can generate a valid commitment for a related value y, then a simulator could perform the same without seeing the commitment C

This is meaningful for perfectly binding commitments only: with an encryption scheme, IND-CCA2 security level guarantees non-malleability

Motivation

ElGamal Commitment

$\mathsf{comEG}_{\mathsf{pk}}(x; r) = \mathsf{EG}^{+}_{\mathsf{pk}}(x; r)$, is extractable for small x only

Example

If $x \in \{0, 1\}$, any $C(x) = \mathsf{comEG}_{\mathsf{pk}}(x; r)$ is extractable

Homomorphic Property

Let us assume $2^{k-1} < q < 2^k$, then for any $x = \sum_{i=0}^{k-1} x_i \times 2^i \in \mathbb{Z}_q$, $C(x) = \{C_i = \mathsf{comEG}_{\mathsf{pk}}(x_i; r_i) = \mathsf{EG}^{+}_{\mathsf{pk}}(x_i; r_i)\}_{i=0}^{k-1}$, is extractable under the condition that $(x_i)_i \in \{0, 1\}^k$

Furthermore, $\mathsf{comEG}_{\mathsf{pk}}(x; r) = \prod C_i^{2^i}$, for $r = \sum_{i=0}^{k-1} r_i \times 2^i$

Extended Languages

$$x = 0 \iff C(x) = \mathsf{comEG}_{\mathsf{pk}}(x; r) \in L_{(\mathbf{EG}^{+}, \rho), 0}$$
$$x = 1 \iff C(x) = \mathsf{comEG}_{\mathsf{pk}}(x; r) \in L_{(\mathbf{EG}^{+}, \rho), 1}$$

We then define

$$L_{(\mathbf{EG}^{+}, \rho), 0 \vee 1} = L_{(\mathbf{EG}^{+}, \rho), 0} \cup L_{(\mathbf{EG}^{+}, \rho), 1}$$

To be extractable, $C = (C_i)_i$ has to lie in

$$L = \{(C_0, \ldots, C_{k-1}) \mid \forall i, C_i \in L_{(\mathbf{EG}^{+}, \rho), 0 \vee 1}\}$$

Certification of Public Keys

For the certification Cert of an ElGamal public key $y = g^x$, in most of the protocols, the simulator needs to be able to extract the secret key:

Classical Process

- the user sends his public key $y = g^x$;
- the user and the authority run a ZK proof of knowledge of x
- if convinced, the authority generates and sends the certificate Cert for y

But for extracting x in the simulation, the reduction requires a rewinding (that is not always allowed: *e.g.*, in the UC Framework)

Certification of Public Keys

For the certification Cert of an ElGamal public key $y = g^x$, in most of the protocols, the simulator needs to be able to extract the secret key:

New Process

the user and the authority use a smooth projective hash system for L:
$\textbf{HASH}(\mathsf{pk}) = (\mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash})$

- the user sends his public key $y = g^x$, together with an L-extractable commitment C of x, with random r;
- the authority generates
  - a hashing key $\mathsf{hk} \stackrel{\$}{\leftarrow} \mathsf{HashKG}()$,
  - the corresponding projected key on C, $\mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, C)$
  - the hash value $\mathsf{Hash} = \mathsf{Hash}(\mathsf{hk}; C)$

  and sends hp along with Cert $\oplus$ Hash;
- The user computes $\mathsf{Hash} = \mathsf{ProjHash}(\mathsf{hp}; C, r)$, and gets Cert.

Commitment and Smooth Projective HF

The authority sends hp along with Cert $\oplus$ Hash

Analysis: Correct Commitment

If the user correctly computed the commitment ($C \in L$)

- he knows the witness r, and can get the same mask Hash;
- the simulator can extract x, granted the L-extractability

Analysis: Incorrect Commitment

If the user cheated ($C \notin L$)

- the simulator is not guaranteed to extract anything;
- but, the smoothness property makes Hash perfectly unpredictable: no information is leaked about the certificate.

Outline

1. Smooth Projective Hash Functions
   - Definitions
   - Conjunctions and Disjunctions
2. Extractable Commitments
   - Properties
   - Conditional Extractability
   - Application: Certification of Public Keys
3. Equivocable and Extractable Commitments
   - Description
   - Analysis
4. Adaptive Security and UC PAKE
   - Universal Composability
   - Previous Schemes
   - Our Scheme

A First Approach

[Canetti-Fischlin C '01]

To get both extractability and equivocability (at the same time), one can combine perfectly hiding and perfectly binding commitments:

- Pedersen's commitment is perfectly hiding
- ElGamal's commitment is perfectly binding

Notations

if b is a bit, we denote its complement by $\overline{b}$
$x[i]$ denotes the i^{th} bit of the bit-string x

Extractable and Equivocable Commitment

Common Reference String Model

The commitment is realized in the common reference string model: the CRS ρ contains

- (G, pk), where pk is an ElGamal public key and the private key is unknown to anybody (except to the commitment extractor)
- the tuple $(y_1, \ldots, y_m) \in G^m$, for which the discrete logarithms in basis g are unknown to anybody (except to the commitment equivocator)

Let the input of the committing algorithm be a bit-string

$$\pi = \sum_{i=1}^{m} \pi_i \cdot 2^{i-1}$$

Extractable and Equivocable Commitment

In order to commit to π, for $i = 1, \ldots, m$,

- one chooses a random value $x_{i,\pi_i} = \sum_{j=1}^{n} x_{i,\pi_i}[j] \cdot 2^{j-1} \in \mathbb{Z}_q$ and sets $x_{i,\overline{\pi}_i} = 0$
- one commits to π_i, using the random x_{i,π_i}:

$$a_i = \mathsf{comPed}(\pi_i, x_{i,\pi_i}) = g^{x_{i,\pi_i}} y_i^{\pi_i}$$

  This defines $\mathbf{a} = (a_1, \ldots, a_m)$
- one commits to $x_{i,\delta}$, for $\delta = 0, 1$: $(\mathbf{b}_{i,\delta} = (b_{i,\delta}[j])_j = \mathsf{comEG}_{\mathsf{pk}}(x_{i,\delta})$, where $b_{i,\delta}[j] = \mathsf{EG}^+_{\mathsf{pk}}(x_{i,\delta}[j] \cdot 2^{j-1}, r_{i,\delta}[j])$

Then, $B_{i,\delta} = \prod_j b_{i,\delta}[j] = \mathsf{EG}^+_{\mathsf{pk}}(x_{i,\delta}, r_{i,\delta})$, where $r_{i,\delta} = \sum_j r_{i,\delta}[j]$.

Extractable and Equivocable Commitment

- Random string:

$$R = (x_{1,\pi_1}, (r_{1,0}[j], r_{1,1}[j])_j, \ldots, x_{m,\pi m}, (r_{m,0}[j], r_{m,1}[j])_j)$$

- Commitment: $\mathsf{com}_\rho(\pi; R) = (\mathbf{a}, \mathbf{b})$

$$\text{where}\quad \mathbf{a} = (a_i = \mathsf{comPed}(\pi_i, x_{i,\pi_i}))_i$$
$$\mathbf{b} = (b_{i,\delta}[j] = \mathsf{EG}^+_{\mathsf{pk}}(x_{i,\delta}[j] \cdot 2^{j-1}, r_{i,\delta}[j]))_{i,\delta,j}$$

- Witness: the values $r_{i,\overline{\pi}_i}[j]$ can be erased,

$$\mathsf{w} = (x_{1,\pi_1}, (r_{1,\pi_1}[j])_j, \ldots, x_{m,\pi m}, (r_{m,\pi m}[j])_j)$$

- Opening: given the above witness, and the value π

$$\forall i, j : b_{i,\pi_i}[j] \stackrel{?}{=} \mathsf{EG}^+_{\mathsf{pk}}(x_{i,\pi_i}[j] \cdot 2^{j-1}, r_{i,\pi_i}[j])$$
$$\forall i : a_i \stackrel{?}{=} \mathsf{comPed}(\pi_i, x_{i,\pi_i})$$

Properties

$$\mathsf{com}_\rho(\pi; R) = (\mathbf{a}, \mathbf{b}) : \quad \begin{aligned} \mathbf{a} &= (a_i = \mathsf{comPed}(\pi_i, x_{i,\pi_i}))_i \\ \mathbf{b} &= (b_{i,\delta}[j] = \mathsf{EG}^+_{\mathsf{pk}}(x_{i,\delta}[j]\cdot 2^{j-1}, r_{i,\delta}[j]))_{i,\delta,j} \end{aligned}$$

Intuition

- Granted the perfectly hiding property of the Pedersen commitment, without any information on the $x_{i,\delta}[j]$'s, no information is leaked about the π_i's
- Granted the semantic security of the ElGamal encryption scheme, the former privacy on the $x_{i,\delta}[j]$'s is guaranteed
- Granted the computationally binding property of the Pedersen commitment, the a_i's cannot be open in two ways

Conditional Extractability

Constraints

- bit-by-bit encryption of the $x_{i,\delta}[j]$:
  with the ElGamal decryption key, one decrypts all the $b_{i,\delta}[j]$, and gets the x_{i,π_i} (unless the plaintexts are different to 0 and 2^{i-1})
- then, one can confirm, for $i = 1, \ldots, m$, whether $a_i = \mathsf{comPed}(0, x_{i,0})$ or $a_i = \mathsf{comPed}(1, x_{i,1})$, which provides π_i (unless none of the equalities is satisfied)

The above conditions define the language for extractability:

$$L_{\rho,\pi} = \left\{ C \;\middle|\; \begin{array}{l} \exists R \text{ such that } C = \mathsf{com}_\rho(\pi, R) \\ \text{and } \forall i\ \forall j\ \ b_{i,\pi_i}[j] \in L_{(\mathbf{EG}^+,\rho),0\vee 1} \\ \text{and } \forall i\ \ B_{i,\pi_i} \in L_{(\mathbf{EG}^\times,\rho),a_i/y_i^{\pi_i}} \end{array} \right\}$$

Equivocability

Normal Procedure

- One takes a random x_{i,π_i} and then $x_{i,\overline{\pi}_i} = 0$, which specifies π_i
- One commits on π_i using randomness x_{i,π_i}
- One encrypts both x_{i,π_i} and $x_{i,\overline{\pi}_i}$, bit-by-bit

Equivocable Procedure

Granted the Pedersen commitment trapdoor

- one takes a random $x_{i,0}$ and extracts $x_{i,1}$ such that $a_i = \mathsf{comPed}(0, x_{i,0}) = \mathsf{comPed}(1, x_{i,1})$
- the rest of the commitment procedure remains the same

One can open any bit-string for π, using the appropriate x_{i,π_i} and the corresponding random elements (no erasure)

Non-Malleability

Using a non-malleable encryption scheme (Cramer-Shoup), one can make the commitment non-malleable:

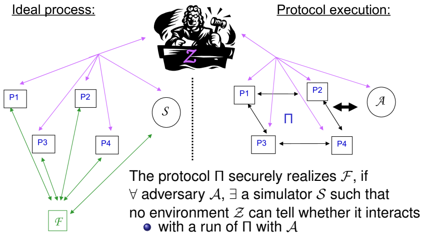
- Random string:
  $$R = (x_{1,\pi_1}, (r_{1,0}[j], r_{1,1}[j])_j, \ldots, x_{m,\pi_m}, (r_{m,0}[j], r_{m,1}[j])_j)$$
- Commitment: $\mathsf{com}_\rho(\pi; R) = (\mathbf{a}, \mathbf{b})$
  $$\text{where} \quad \begin{aligned} \mathbf{a} &= (a_i = \mathsf{comPed}(\pi_i, x_{i,\pi_i}))_i \\ \mathbf{b} &= (b_{i,\delta}[j] = \mathsf{CS}^+_{\mathsf{pk}}(x_{i,\delta}[j]\cdot 2^{j-1}, r_{i,\delta}[j]))_{i,\delta,j} \end{aligned}$$
- Opening: given the above witness, and the value π
  $$\begin{aligned} \forall i,j : b_{i,\pi_i}[j] &\stackrel{?}{=} \mathsf{CS}^+_{\mathsf{pk}}(x_{i,\pi_i}[j]\cdot 2^{j-1}, r_{i,\pi_i}[j]) \\ \forall i : a_i &\stackrel{?}{=} \mathsf{comPed}(\pi_i, x_{i,\pi_i}) \end{aligned}$$

Outline

1. Smooth Projective Hash Functions
   - Definitions
   - Conjunctions and Disjunctions
2. Extractable Commitments
   - Properties
   - Conditional Extractability
   - Application: Certification of Public Keys
3. Equivocable and Extractable Commitments
   - Description
   - Analysis
4. **Adaptive Security and UC PAKE**
   - Universal Composability
   - Previous Schemes
   - Our Scheme

Universal Composability

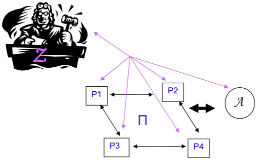
[Canetti FOCS '01]



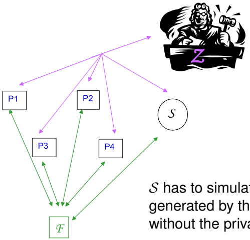
The protocol Π securely realizes $\mathcal{F}$, if $\forall$ adversary $\mathcal{A}$, $\exists$ a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell whether it interacts

- with a run of Π with $\mathcal{A}$
- or with an ideal run with $\mathcal{F}$ and $\mathcal{S}$

Real-life Execution



Ideal Execution



$\mathcal{S}$ has to simulate the view generated by the honest users without the private inputs

Password-Authenticated Key Exchange

Definition

Two players want to establish a common secret key, using a short password as authentication means: exhaustive search is possible

- on-line dictionary attack: Elimination of one candidate per attack. This is unavoidable
- off-line dictionary attack: the transcript of a communication helps to eliminate one or a few candidates
  This is avoidable, and should be avoided

One wants to prove that eliminating one candidate per active attempt is the best attack

Ideal Functionality

[Canetti-Halevi-Katz-Lindell-MacKenzie EC ’05]

Functionality $\mathcal{F}_{\mathsf{pwKE}}$

The functionality $\mathcal{F}_{\mathsf{pwKE}}$ is parameterized by a security parameter k. It interacts with an adversary S and a set of parties via the following queries:

Upon receiving a query $(\mathsf{NewSession}, sid, P_i, P_j, pw, \mathrm{role})$ **from party** P_i:
Send $(\mathsf{NewSession}, sid, P_i, P_j, \mathrm{role})$ to S. In addition, if this is the first NewSession query, or if this is the second NewSession query and there is a record (P_j, P_i, pw'), then record (P_i, P_j, pw) and mark this record fresh.

Upon receiving a query $(\mathsf{TestPwd}, sid, P_i, pw')$ **from the adversary** S:
If there is a record of the form (P_i, P_j, pw) which is fresh, then do: If $pw = pw'$, mark the record compromised and reply to S with “correct guess”. If $pw \neq pw'$, mark the record interrupted and reply with “wrong guess”.

Upon receiving a query $(\mathsf{NewKey}, sid, P_i, sk)$ **from** S**, where** $|sk| = k$:
If there is a record of the form (P_i, P_j, pw), and this is the first NewKey query for P_i, then:

- If this record is compromised, or either P_i or P_j is corrupted, then output (sid, sk) to player P_i.
- If this record is fresh, and there is a record (P_j, P_i, pw') with $pw' = pw$, and a key sk' was sent to P_j, and (P_j, P_i, pw) was fresh at the time, then output (sid, sk') to P_i.
- In any other case, pick a new random key sk' of length k and send (sid, sk') to P_i.

Either way, mark the record (P_i, P_j, pw) as completed.

Figure 2: The password-based key-exchange functionality $\mathcal{F}_{\mathsf{pwKE}}$

TestPwd to model on-line dictionary attacks (once per session)

Scheme I

[Katz-Ostrovsky-Yung EC ’01, Gennaro-Lindell C ’03]

P_i (client)	CRS: pke	P_j (server)
	$\xleftarrow{c_1, vk}$	$(sk, vk) \leftarrow \mathrm{sigKey}(\$)$ $c_1 \leftarrow E_{pke}(pw; r_1)$
$c_2 \leftarrow E_{pke}(pw, r_2)$ $hk \leftarrow \mathcal{H}$ $hp \leftarrow \alpha(hk; c_1)$	$\xrightarrow{c_2, hp}$	
	$\xleftarrow{hp', \sigma}$	$hk' \leftarrow \mathcal{H}$ $hp' \leftarrow \alpha(hk'; c_2)$ $\sigma \leftarrow \mathsf{Sign}_{sk}(c_2, hp, hp')$
if $(\mathsf{Verify}_{vk}((c_2, hp, hp'), \sigma) = 1)$ session-key $\leftarrow H_{hk}(c_1, pw)$ $+ h_{hp'}(c_2, pw; r_2)$		session-key $\leftarrow h_{hp}(c_1, pw; r_1)$ $+ H_{hk'}(c_2, pw)$

Analysis

Security in the classical framework:

- Commitment to an incorrect password: smoothness leads to a perfectly random session key
- Replay of a commitment: pseudo-randomness leads to a computationally random session key (witness unknown)

P_i (client)	CRS: pke	P_j (server)
	$\xleftarrow{c_1, vk}$	$(sk, vk) \leftarrow \mathrm{sigKey}(\$)$ $c_1 \leftarrow E_{pke}(pw; r_1)$
$c_2 \leftarrow E_{pke}(pw, r_2)$ $hk \leftarrow \mathcal{H}$ $hp \leftarrow \alpha(hk; c_1)$	$\xrightarrow{c_2, hp}$	
	$\xleftarrow{hp', \sigma}$	$hk' \leftarrow \mathcal{H}$ $hp' \leftarrow \alpha(hk'; c_2)$ $\sigma \leftarrow \mathsf{Sign}_{sk}(c_2, hp, hp')$
if $(\mathsf{Verify}_{vk}((c_2, hp, hp'), \sigma) = 1)$ session-key $\leftarrow H_{hk}(c_1, pw)$ $+ h_{hp'}(c_2, pw; r_2)$		session-key $\leftarrow h_{hp}(c_1, pw; r_1)$ $+ H_{hk'}(c_2, pw)$

Simulation of the honest players: use of a dummy password

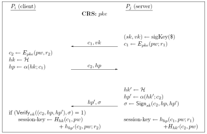
- indistinguishable, unless $\mathcal{A}$ committed to the correct password: $\mathcal{S}$ cannot compute the correct key $\Longrightarrow \mathcal{S}$ aborts
- in the UC framework, $\mathcal{Z}$ sees the difference between a real-execution and the simulation: when $\mathcal{A}$ wins, $\mathcal{S}$ aborts
  Because of the short password, this is not negligible

Analysis

If $\mathcal{A}$ plays the server role:

- $\mathcal{S}$ can extract the committed password, and check it granted the TestPwd query
- password valid: $\mathcal{S}$ uses it
- else: dummy password

$\Longrightarrow$ perfect simulation

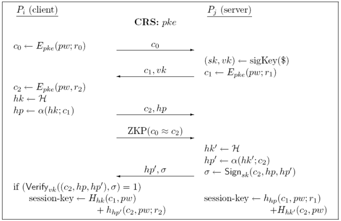


If $\mathcal{A}$ plays the client role:

- $\mathcal{S}$ does not know yet the password sent by $\mathcal{A}$: dummy password
- when $\mathcal{A}$ sends its commitment, $\mathcal{S}$ extracts the password and checks it granted the TestPwd query
- if the password is invalid, $\mathcal{S}$ follows with the dummy password
- else, $\mathcal{S}$ is stuck

Scheme II

[Canetti-Halevi-Katz-Lindell-MacKenzie EC '05]



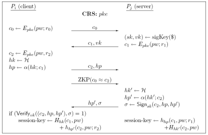
Add of a first commitment round

Analysis

If $\mathcal{A}$ plays the client role:

- $\mathcal{S}$ can extract the committed password, and check it granted the TestPwd query
- password valid: $\mathcal{S}$ uses it
- else: dummy password

$\Longrightarrow$ perfect simulation



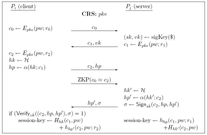
If $\mathcal{A}$ plays the server role:

- $\mathcal{S}$ does not know yet the password: dummy password in c_0
- when $\mathcal{A}$ sends its commitment c_1, $\mathcal{S}$ extracts the password and checks it granted the TestPwd query
- if the password is invalid, $\mathcal{S}$ follows with the dummy password
- else, $\mathcal{S}$ uses the correct password in c_2 and simulates the ZKP

Adaptive Corruption

If $\mathcal{A}$ plays the server role:

- $\mathcal{S}$ does not know the password: dummy password in c_0
- $\mathcal{S}$ extracts the password from c_1 checks it (TestPwd query)
- if invalid: $\mathcal{S}$ follows with the dummy password in c_2
- else, $\mathcal{S}$ uses the correct password in c_2 and simulate the ZKP



What about if $\mathcal{A}$ corrupts the client right after c_0?
$\mathcal{S}$ gets the correct password, but cannot open c_0 correctly!
$\Longrightarrow$ security against static-corruptions only (before the session starts)

Non-malleable, L-extractable, equivocable commitment provides adaptive security

Adaptively Secure UC-PAKE

(U1) $(\mathrm{VK_I}, \mathrm{SK_I}) \leftarrow \mathsf{SKG}$
$\ell_\mathrm{I} = \mathrm{J} \circ \mathrm{I} \circ \mathrm{ssid} \circ \mathrm{VK_I}$
$\mathsf{com_I} = \mathsf{com}_\rho(\ell_\mathrm{I}, \mathsf{pw_I}; R_\mathrm{I})$

$\xrightarrow{(\mathsf{com_I}, \mathrm{VK_I})}$

(S2) (publicly) checks the validity of $\mathsf{com_I}$
$(\mathrm{VK_J}, \mathrm{SK_J}) \leftarrow \mathsf{SKG}$
$\ell_\mathrm{J} = \mathrm{J} \circ \mathrm{I} \circ \mathrm{ssid} \circ \mathrm{VK_J}$
$\mathsf{hk_J} = \mathsf{HashKG}(\rho, (\ell_\mathrm{I}, \mathsf{pw_J}), r_J)$
$\mathsf{com_J} = \mathsf{com}_\rho(\ell_\mathrm{J}, \mathsf{pw_J}; R_\mathrm{J})$
$\mathsf{hp_J} = \mathsf{ProjKG}(\mathsf{hk_J}; \rho, (\ell_\mathrm{I}, \mathsf{pw_J}), \mathsf{com_I})$
$\mathtt{Hash_J} = \mathsf{Hash}(\mathsf{hk_J}; \rho, (\ell_\mathrm{I}, \mathsf{pw_J}), \mathsf{com_I})$
erases $\mathsf{hk_J}$

$\xleftarrow{(\mathsf{com_J}, \mathrm{VK_J}, \mathsf{hp_J})}$

(U3) (publicly) checks the validity of $\mathsf{com_J}$
$\mathsf{hk_I}\mathsf{HashKG}(\rho, (\ell_\mathrm{J}, \mathsf{pw_I}), r_I)$
$\mathsf{hp_I} = \mathsf{ProjKG}(\mathsf{hk_I}; (\ell_\mathrm{J}, \mathsf{pw_I}), \mathsf{com_J})$
$\sigma_\mathrm{I} = \mathsf{Sign}(\mathrm{SK_I}, (\mathsf{com_I}, \mathsf{com_J}, \mathsf{hp_I}, \mathsf{hp_J}))$
$\mathsf{sk_I} = \mathsf{ProjHash}(\mathsf{hp_J}; \rho, (\ell_\mathrm{I}, \mathsf{pw_I}), \mathsf{com_I}; w_\mathrm{I})$
$\quad + \mathsf{Hash}(\mathsf{hk_I}; \rho, (\ell_\mathrm{J}, \mathsf{pw_I}), \mathsf{com_J})$
erases $\mathsf{hk_I}$

$\xrightarrow{(\sigma_\mathrm{I}, \mathsf{hp_I})}$

(S4) aborts if
$\mathsf{Ver}(\mathrm{VK_I}, (\mathsf{com_I}, \mathsf{com_J}, \mathsf{hp_I}, \mathsf{hp_J}), \sigma_\mathrm{I}) = 0$
$\sigma_\mathrm{J} = \mathsf{Sign}(\mathrm{SK_J}, (\mathsf{com_I}, \mathsf{com_J}, \mathsf{hp_I}, \mathsf{hp_J}))$
$sk_\mathrm{J} = \mathsf{ProjHash}(\mathsf{hp_I}; \rho, (\ell_\mathrm{J}, \mathsf{pw_J}), \mathsf{com_J}; w_\mathrm{J})$
$\quad + \mathtt{Hash_J}$
outputs $(\mathrm{sid}, \mathrm{ssid}, sk_\mathrm{J})$
erases everything
sets the session as `accepted`

$\xleftarrow{(\sigma_\mathrm{J})}$

(U5) aborts if
$\mathsf{Ver}(\mathrm{VK_J}, (\mathsf{com_I}, \mathsf{com_J}, \mathsf{hp_I}, \mathsf{hp_J}), \sigma_\mathrm{J}) = 0$
outputs $(\mathrm{sid}, \mathrm{ssid}, sk_\mathrm{I})$
erases everything
sets the session as `accepted`

Conclusion

Smooth Projective Hash Functions for Complex Languages

Various Applications

- in place of some ZK proofs
- conditional secure channels
- adaptive security in UC PAKE