

# Smooth Projective Hash Functions and Security against Adaptive Corruptions

David Pointcheval

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



CosyProofs – Barbizon – France  
April 15th, 2010

# Outline

- 1 Smooth Projective Hash Functions**
  - Definitions
  - Conjunctions and Disjunctions
- 2 Commitments**
  - Properties
  - Conditional Extractability
- 3 Password-Authenticated Key Exchange**

# Smooth Projective Hash Functions

[Cramer-Shoup EC '02]

## Family of Smooth Projective Hash Functions

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  $hp$ :  $H(x) = \text{ProjHash}_L(hp; x, w)$

The former  $\text{Hash}_L$  works for all points in the domain  $X$ ,  
the latter  $\text{ProjHash}_L$  works for  $x \in L$  only, and requires a witness  $w$ .

Public mapping  $hk \mapsto hp = \text{ProjKG}_L(hk)$  [Cramer-Shoup EC '02]

Element-based mapping  $hk \mapsto hp = \text{ProjKG}_L(hk; x)$  [Gennaro-Lindell EC '03]

# Smooth Projective Hash Functions

[Cramer-Shoup EC '02]

## Family of Smooth Projective Hash Functions

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  $hp$ :  $H(x) = \text{ProjHash}_L(hp; x, w)$

The former  $\text{Hash}_L$  works for all points in the domain  $X$ ,  
the latter  $\text{ProjHash}_L$  works for  $x \in L$  only, and requires a witness  $w$ .

Public mapping  $hk \mapsto hp = \text{ProjKG}_L(hk)$  [Cramer-Shoup EC '02]

Element-based mapping  $hk \mapsto hp = \text{ProjKG}_L(hk; x)$  [Gennaro-Lindell EC '03]

# Smooth Projective Hash Functions

[Cramer-Shoup EC '02]

## Family of Smooth Projective Hash Functions

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  $hp$ :  $H(x) = \text{ProjHash}_L(hp; x, w)$

The former  $\text{Hash}_L$  works for all points in the domain  $X$ ,  
the latter  $\text{ProjHash}_L$  works for  $x \in L$  only, and requires a witness  $w$ .

Public mapping  $hk \mapsto hp = \text{ProjKG}_L(hk)$  [Cramer-Shoup EC '02]

Element-based mapping  $hk \mapsto hp = \text{ProjKG}_L(hk; x)$  [Gennaro-Lindell EC '03]

# 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  $x \in L$

## Smoothness

For any  $x \notin L$ ,  $H(x)$  and  $\text{hp}$  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  $X \setminus L$

# 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  $x \in L$

## Smoothness

For any  $x \notin L$ ,  $H(x)$  and  $\text{hp}$  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  $X \setminus L$

# 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  $x \in L$

## Smoothness

For any  $x \notin L$ ,  $H(x)$  and  $\text{hp}$  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  $X \setminus L$

# 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  $x \in L$

## Smoothness

For any  $x \notin L$ ,  $H(x)$  and  $\text{hp}$  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  $X \setminus L$

# Example

## Encryption

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

## Example

ElGamal Encryption:  $pk = h = g^x$ , where  $sk = x \xleftarrow{\$} \mathbb{Z}_q$

$$EG_{pk}^{\times}(M; r) = (g^r, h^r M) \quad EG_{pk}^{+}(M; r) = (g^r, h^r g^M)$$

# Notations

Let  $\text{SHS}_1$  and  $\text{SHS}_2$  be two smooth hash systems onto a group  $(G, \oplus)$ , corresponding to the languages  $L_1$  and  $L_2$  respectively:

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

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

$$\text{hk}_1 = \text{HashKG}_1(r_1)$$

$$\text{hp}_1 = \text{ProjKG}_1(\text{hk}_1; c)$$

$$\text{hk}_2 = \text{HashKG}_2(r_2)$$

$$\text{hp}_2 = \text{ProjKG}_2(\text{hk}_2; c)$$

We want to define  $\text{SHS}_\cap$  and  $\text{SHS}_\cup$  for:

$$L_\cap = L_1 \cap L_2$$

$$L_\cup = L_1 \cup L_2$$

# Notations

Let  $\text{SHS}_1$  and  $\text{SHS}_2$  be two smooth hash systems onto a group  $(G, \oplus)$ , corresponding to the languages  $L_1$  and  $L_2$  respectively:

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

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

$$\text{hk}_1 = \text{HashKG}_1(r_1)$$

$$\text{hp}_1 = \text{ProjKG}_1(\text{hk}_1; c)$$

$$\text{hk}_2 = \text{HashKG}_2(r_2)$$

$$\text{hp}_2 = \text{ProjKG}_2(\text{hk}_2; c)$$

We want to define  $\text{SHS}_\cap$  and  $\text{SHS}_\cup$  for:

$$L_\cap = L_1 \cap L_2$$

$$L_\cup = L_1 \cup L_2$$

# Notations

Let  $\text{SHS}_1$  and  $\text{SHS}_2$  be two smooth hash systems onto a group  $(G, \oplus)$ , corresponding to the languages  $L_1$  and  $L_2$  respectively:

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

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

$$\text{hk}_1 = \text{HashKG}_1(r_1) \qquad \text{hp}_1 = \text{ProjKG}_1(\text{hk}_1; c)$$

$$\text{hk}_2 = \text{HashKG}_2(r_2) \qquad \text{hp}_2 = \text{ProjKG}_2(\text{hk}_2; c)$$

We want to define  $\text{SHS}_\cap$  and  $\text{SHS}_\cup$  for:

$$L_\cap = L_1 \cap L_2 \qquad L_\cup = L_1 \cup L_2$$

# Conjunction of Languages

If  $c \in L_{\cap} = L_1 \cap L_2$  and  $w_i$  is a witness that  $c \in L_i$ , for  $i = 1, 2$ :

$$\text{HashKG}_{L_{\cap}}(r = r_1 \| r_2) = \text{hk} = (\text{hk}_1, \text{hk}_2)$$

$$\text{ProjKG}_{L_{\cap}}(\text{hk}; c) = \text{hp} = (\text{hp}_1, \text{hp}_2)$$

$$\text{Hash}_{L_{\cap}}(\text{hk}; c) = \text{Hash}_1(\text{hk}_1; c) \oplus \text{Hash}_2(\text{hk}_2; c)$$

$$\text{ProjHash}_{L_{\cap}}(\text{hp}; c, (w_1, w_2)) = \text{ProjHash}_1(\text{hp}_1; c, w_1) \\ \oplus \text{ProjHash}_2(\text{hp}_2; c, w_2)$$

- 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**

# Conjunction of Languages

If  $c \in L_{\cap} = L_1 \cap L_2$  and  $w_i$  is a witness that  $c \in L_i$ , for  $i = 1, 2$ :

$$\text{HashKG}_{L_{\cap}}(r = r_1 \| r_2) = \text{hk} = (\text{hk}_1, \text{hk}_2)$$

$$\text{ProjKG}_{L_{\cap}}(\text{hk}; c) = \text{hp} = (\text{hp}_1, \text{hp}_2)$$

$$\text{Hash}_{L_{\cap}}(\text{hk}; c) = \text{Hash}_1(\text{hk}_1; c) \oplus \text{Hash}_2(\text{hk}_2; c)$$

$$\text{ProjHash}_{L_{\cap}}(\text{hp}; c, (w_1, w_2)) = \text{ProjHash}_1(\text{hp}_1; c, w_1)$$

$$\oplus \text{ProjHash}_2(\text{hp}_2; c, w_2)$$

- 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

If  $c \in L_U = L_1 \cup L_2$  and  $w$  is a witness that  $c \in L_i$  for  $i \in \{1, 2\}$ :

$$\text{HashKG}_{L_U}(r = r_1 \| r_2) = \text{hk} = (\text{hk}_1, \text{hk}_2)$$

$$\text{ProjKG}_{L_U}(\text{hk}; c) = \text{hp} = (\text{hp}_1, \text{hp}_2, \text{hp}_\Delta)$$

$$\text{where } \text{hp}_\Delta = \text{Hash}_1(\text{hk}_1; c) \oplus \text{Hash}_2(\text{hk}_2; c)$$

$$\text{Hash}_{L_U}(\text{hk}; c) = \text{Hash}_1(\text{hk}_1; c)$$

$$\text{ProjHash}_{L_U}(\text{hp}; c, w) = \text{ProjHash}_1(\text{hp}_1; c, w) \text{ if } c \in L_1$$

$$\text{or } \text{hp}_\Delta \ominus \text{ProjHash}_2(\text{hp}_2; c, w)$$

$$\text{if } c \in L_2$$

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

# Disjunction of Languages

If  $c \in L_U = L_1 \cup L_2$  and  $w$  is a witness that  $c \in L_i$  for  $i \in \{1, 2\}$ :

$$\text{HashKG}_{L_U}(r = r_1 \| r_2) = \text{hk} = (\text{hk}_1, \text{hk}_2)$$

$$\text{ProjKG}_{L_U}(\text{hk}; c) = \text{hp} = (\text{hp}_1, \text{hp}_2, \text{hp}_\Delta)$$

$$\text{where } \text{hp}_\Delta = \text{Hash}_1(\text{hk}_1; c) \oplus \text{Hash}_2(\text{hk}_2; c)$$

$$\text{Hash}_{L_U}(\text{hk}; c) = \text{Hash}_1(\text{hk}_1; c)$$

$$\text{ProjHash}_{L_U}(\text{hp}; c, w) = \text{ProjHash}_1(\text{hp}_1; c, w) \text{ if } c \in L_1$$

$$\text{or } \text{hp}_\Delta \ominus \text{ProjHash}_2(\text{hp}_2; c, w)$$

$$\text{if } c \in L_2$$

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

# Commitments

## Definition

A commitment scheme is defined by two algorithms:

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

## Properties

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

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

# Commitments

## Definition

A commitment scheme is defined by two algorithms:

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

## Properties

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

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

# Examples

Let us be given  $(G, q, g, pk = h = g^x)$

## ElGamal

- $C = \text{comEG}_{pk}(m; r) = \text{EG}_{pk}^+(m; r) = (g^r, h^r g^m)$  with  $r \xleftarrow{\$} \mathbb{Z}_q$ ;
- **perfectly binding** and **computationally hiding**, (DDH assumption)

## Pedersen

- $C = \text{comPed}(m; r) = g^m h^r$ , with  $r \xleftarrow{\$} \mathbb{Z}_q$ ;
- **perfectly hiding** and **computationally binding**, (DL assumption)

# Examples

Let us be given  $(G, q, g, pk = h = g^x)$

## EIGamal

- $C = \text{comEG}_{pk}(m; r) = \text{EG}_{pk}^+(m; r) = (g^r, h^r g^m)$  with  $r \xleftarrow{\$} \mathbb{Z}_q$ ;
- **perfectly binding** and **computationally hiding**, (DDH assumption)

## Pedersen

- $C = \text{comPed}(m; r) = g^m h^r$ , with  $r \xleftarrow{\$} \mathbb{Z}_q$ ;
- **perfectly hiding** and **computationally binding**, (DL assumption)

# Examples

Let us be given  $(G, q, g, pk = h = g^x)$

## EIGamal

- $C = \text{comEG}_{pk}(m; r) = \text{EG}_{pk}^+(m; r) = (g^r, h^r g^m)$  with  $r \xleftarrow{\$} \mathbb{Z}_q$ ;
- **perfectly binding** and **computationally hiding**, (DDH assumption)

## Pedersen

- $C = \text{comPed}(m; r) = g^m h^r$ , with  $r \xleftarrow{\$} \mathbb{Z}_q$ ;
- **perfectly hiding** and **computationally binding**, (DL assumption)

# Additional Properties

## Extractability (if computationally hiding)

An **extractor** can extract  $m$  from any  $C = \text{com}(m, r)$  (trapdoor)

E.g. an encryption scheme, the decryption key is the extraction key

## Equivocability (if computationally binding)

An **equivocator** can open a commitment in different ways (trapdoor)

## Non-Malleability (if computationally hiding)

From a commitment  $C$  of some unknown value  $m$ , it is hard to generate a valid commitment for a related value  $m'$

E.g. an IND-CCA2 encryption scheme

# Additional Properties

## Extractability (if computationally hiding)

An **extractor** can extract  $m$  from any  $C = \text{com}(m, r)$  (trapdoor)

E.g. an encryption scheme, the decryption key is the extraction key

## Equivocability (if computationally binding)

An **equivocator** can open a commitment in different ways (trapdoor)

## Non-Malleability (if computationally hiding)

From a commitment  $C$  of some unknown value  $m$ , it is hard to generate a valid commitment for a related value  $m'$

E.g. an IND-CCA2 encryption scheme

# Additional Properties

## Extractability (if computationally hiding)

An **extractor** can extract  $m$  from any  $C = \text{com}(m, r)$  (trapdoor)

E.g. an encryption scheme, the decryption key is the extraction key

## Equivocability (if computationally binding)

An **equivocator** can open a commitment in different ways (trapdoor)

## Non-Malleability (if computationally hiding)

From a commitment  $C$  of some unknown value  $m$ , it is hard to generate a valid commitment for a related value  $m'$

E.g. an IND-CCA2 encryption scheme

# Additional Properties

## Extractability (if computationally hiding)

An **extractor** can extract  $m$  from any  $C = \text{com}(m, r)$  (trapdoor)

E.g. an encryption scheme, the decryption key is the extraction key

## Equivocability (if computationally binding)

An **equivocator** can open a commitment in different ways (trapdoor)

## Non-Malleability (if computationally hiding)

From a commitment  $C$  of some unknown value  $m$ , it is hard to generate a valid commitment for a related value  $m'$

E.g. an IND-CCA2 encryption scheme

# Additional Properties

## Extractability (if computationally hiding)

An **extractor** can extract  $m$  from any  $C = \text{com}(m, r)$  (trapdoor)

E.g. an encryption scheme, the decryption key is the extraction key

## Equivocability (if computationally binding)

An **equivocator** can open a commitment in different ways (trapdoor)

## Non-Malleability (if computationally hiding)

From a commitment  $C$  of some unknown value  $m$ , it is hard to generate a valid commitment for a related value  $m'$

E.g. an IND-CCA2 encryption scheme

# Additional Properties

## Extractability (if computationally hiding)

An **extractor** can extract  $m$  from any  $C = \text{com}(m, r)$  (trapdoor)

E.g. an encryption scheme, the decryption key is the extraction key

## Equivocability (if computationally binding)

An **equivocator** can open a commitment in different ways (trapdoor)

## Non-Malleability (if computationally hiding)

From a commitment  $C$  of some unknown value  $m$ , it is hard to generate a valid commitment for a related value  $m'$

E.g. an IND-CCA2 encryption scheme

# Motivation

## ElGamal Commitment

$\text{comEG}_{\text{pk}}(m; r) = \text{EG}_{\text{pk}}^+(m; r) = (g^r, h^r g^m)$ ,  
is extractable for small  $m$  only

## Example

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

## Homomorphic Property

If  $2^{k-1} < q$ , then for any  $m = \sum_{i=0}^{k-1} m_i \times 2^i \in \mathbb{Z}_q$ ,

$$C(m) = \{C_i = \text{comEG}_{\text{pk}}(m_i; r_i) = \text{EG}_{\text{pk}}^+(m_i; r_i)\}_{i=0}^{k-1},$$

is extractable under the condition that  $(m_i)_i \in \{0, 1\}^k$

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

# Motivation

## ElGamal Commitment

$\text{comEG}_{\text{pk}}(m; r) = \text{EG}_{\text{pk}}^+(m; r) = (g^r, h^r g^m)$ ,  
is extractable for small  $m$  only

## Example

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

## Homomorphic Property

If  $2^{k-1} < q$ , then for any  $m = \sum_{i=0}^{k-1} m_i \times 2^i \in \mathbb{Z}_q$ ,

$$C(m) = \{C_i = \text{comEG}_{\text{pk}}(m_i; r_i) = \text{EG}_{\text{pk}}^+(m_i; r_i)\}_{i=0}^{k-1},$$

is extractable under the condition that  $(m_i)_i \in \{0, 1\}^k$

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

# Motivation

## ElGamal Commitment

$\text{comEG}_{\text{pk}}(m; r) = \text{EG}_{\text{pk}}^+(m; r) = (g^r, h^r g^m)$ ,  
is extractable for small  $m$  only

## Example

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

## Homomorphic Property

If  $2^{k-1} < q$ , then for any  $m = \sum_{i=0}^{k-1} m_i \times 2^i \in \mathbb{Z}_q$ ,

$$C(m) = \{C_i = \text{comEG}_{\text{pk}}(m_i; r_i) = \text{EG}_{\text{pk}}^+(m_i; r_i)\}_{i=0}^{k-1},$$

is extractable under the condition that  $(m_i)_i \in \{0, 1\}^k$

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

# Extended Languages

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

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

We then define

$$L_{\text{EG}^+, 0 \vee 1} = L_{\text{EG}^+, 0} \cup L_{\text{EG}^+, 1}$$

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

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

A conjunction of disjunctions

# Extended Languages

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

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

We then define

$$L_{\text{EG}^+, 0 \vee 1} = L_{\text{EG}^+, 0} \cup L_{\text{EG}^+, 1}$$

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

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

A conjunction of disjunctions

# Extended Languages

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

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

We then define

$$L_{\text{EG}^+, 0 \vee 1} = L_{\text{EG}^+, 0} \cup L_{\text{EG}^+, 1}$$

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

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

**A conjunction of disjunctions**

# 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

# 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

# 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

# Password-Authenticated Key Exchange

## Definition

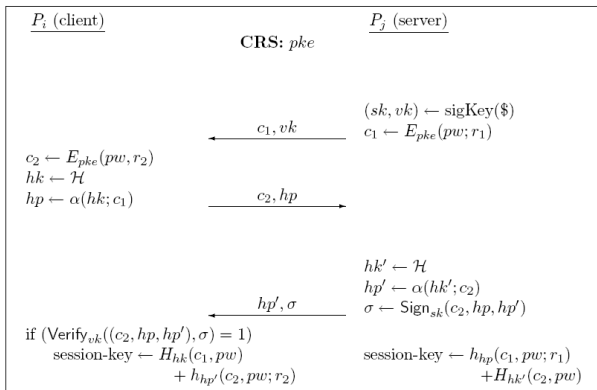
Two players want to establish a common secret key, using a short password as authentication means: exhaustive search is possible

- 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

## Scheme I

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



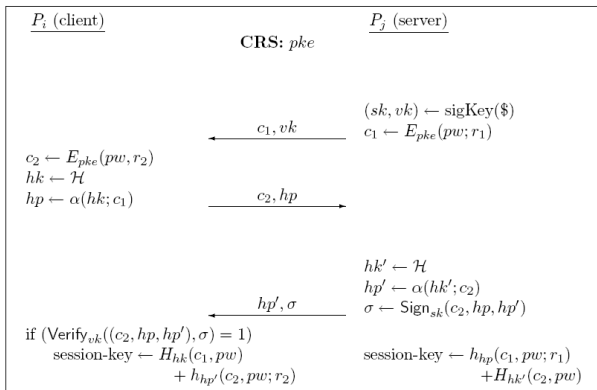
Not UC secure!

Add of a first commitment round: for non-adaptive UC security

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

## Scheme I

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



Not UC secure!

Add of a first commitment round: for non-adaptive UC security

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

# Scheme II

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

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

- $\mathcal{S}$  can extract the committed password in  $c_0$ , and check it

⇒ perfect simulation of  $c_1$

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

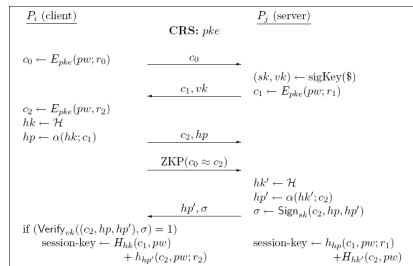
- $\mathcal{S}$  uses dummy password in  $c_0$
- From  $c_1$ ,  $\mathcal{S}$  extracts the committed password, and checks it

⇒ perfect simulation of  $c_2$  and ZKP

What about if  $\mathcal{A}$  corrupts the client right after  $c_0$ ?

⇒ security against static-corruptions only (before the session starts)

Non-malleable,  $L$ -extractable, equivocable commitment provides adaptive security to the KOY/GL construction



## Scheme II

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

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

- $\mathcal{S}$  can extract the committed password in  $c_0$ , and check it

⇒ perfect simulation of  $c_1$

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

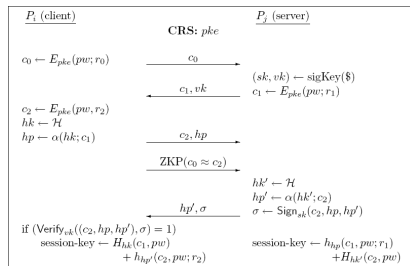
- $\mathcal{S}$  uses dummy password in  $c_0$
- From  $c_1$ ,  $\mathcal{S}$  extracts the committed password, and checks it

⇒ perfect simulation of  $c_2$  and ZKP

What about if  $\mathcal{A}$  corrupts the client right after  $c_0$ ?

⇒ security against static-corruptions only (before the session starts)

Non-malleable,  $L$ -extractable, equivocable commitment provides adaptive security to the KOY/GL construction



# Scheme II

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

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

- $\mathcal{S}$  can extract the committed password in  $c_0$ , and check it

$\implies$  perfect simulation of  $c_1$

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

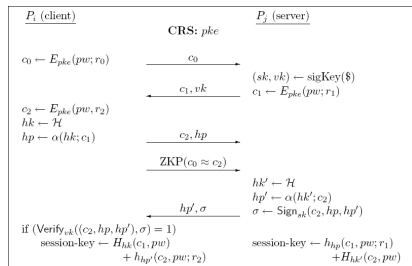
- $\mathcal{S}$  uses dummy password in  $c_0$
- From  $c_1$ ,  $\mathcal{S}$  extracts the committed password, and checks it

$\implies$  perfect simulation of  $c_2$  and ZKP

What about if  $\mathcal{A}$  corrupts the client right after  $c_0$ ?

$\implies$  security against static-corruptions only (before the session starts)

Non-malleable,  $L$ -extractable, equivocable commitment provides adaptive security to the KOY/GL construction



# Scheme II

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

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

- $\mathcal{S}$  can extract the committed password in  $c_0$ , and check it

⇒ perfect simulation of  $c_1$

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

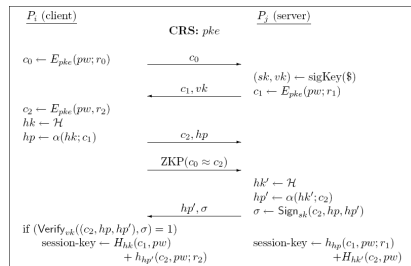
- $\mathcal{S}$  uses dummy password in  $c_0$
- From  $c_1$ ,  $\mathcal{S}$  extracts the committed password, and checks it

⇒ perfect simulation of  $c_2$  and ZKP

What about if  $\mathcal{A}$  corrupts the client right after  $c_0$ ?

⇒ security against static-corruptions only (before the session starts)

Non-malleable,  $L$ -extractable, equivocable commitment provides adaptive security to the KOY/GL construction



## Scheme II

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

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

- $\mathcal{S}$  can extract the committed password in  $c_0$ , and check it

⇒ perfect simulation of  $c_1$

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

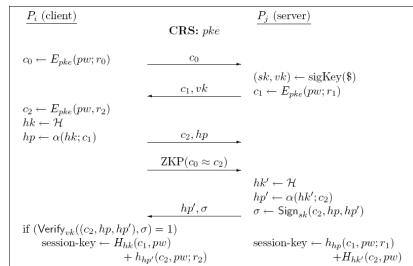
- $\mathcal{S}$  uses dummy password in  $c_0$
- From  $c_1$ ,  $\mathcal{S}$  extracts the committed password, and checks it

⇒ perfect simulation of  $c_2$  and ZKP

What about if  $\mathcal{A}$  corrupts the client right after  $c_0$ ?

⇒ security against static-corruptions only (before the session starts)

Non-malleable,  $L$ -extractable, equivocable commitment provides adaptive security to the KOY/GL construction



# Scheme II

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

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

- $\mathcal{S}$  can extract the committed password in  $c_0$ , and check it

⇒ perfect simulation of  $c_1$

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

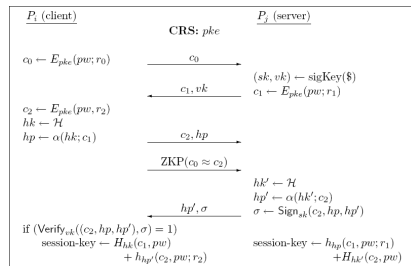
- $\mathcal{S}$  uses dummy password in  $c_0$
- From  $c_1$ ,  $\mathcal{S}$  extracts the committed password, and checks it

⇒ perfect simulation of  $c_2$  and ZKP

What about if  $\mathcal{A}$  corrupts the client right after  $c_0$ ?

⇒ security against static-corruptions only (before the session starts)

Non-malleable,  $L$ -extractable, equivocable commitment provides adaptive security to the KOY/GL construction



# Scheme II

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

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

- $\mathcal{S}$  can extract the committed password in  $c_0$ , and check it

⇒ perfect simulation of  $c_1$

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

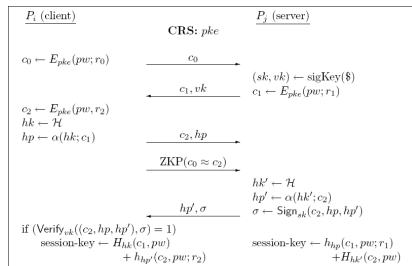
- $\mathcal{S}$  uses dummy password in  $c_0$
- From  $c_1$ ,  $\mathcal{S}$  extracts the committed password, and checks it

⇒ perfect simulation of  $c_2$  and ZKP

What about if  $\mathcal{A}$  corrupts the client right after  $c_0$ ?

⇒ security against static-corruptions only (before the session starts)

Non-malleable,  $L$ -extractable, equivocable commitment provides adaptive security to the KOY/GL construction



## Scheme II

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

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

- $\mathcal{S}$  can extract the committed password in  $c_0$ , and check it

⇒ perfect simulation of  $c_1$

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

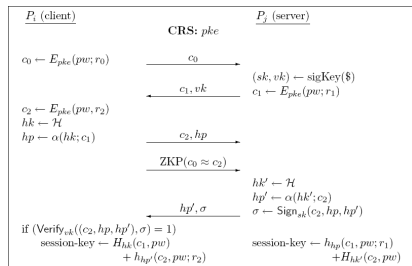
- $\mathcal{S}$  uses dummy password in  $c_0$
- From  $c_1$ ,  $\mathcal{S}$  extracts the committed password, and checks it

⇒ perfect simulation of  $c_2$  and ZKP

What about if  $\mathcal{A}$  corrupts the client right after  $c_0$ ?

⇒ security against static-corruptions only (before the session starts)

Non-malleable,  $L$ -extractable, equivocable commitment provides adaptive security to the KOY/GL construction



# Scheme II

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

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

- $\mathcal{S}$  can extract the committed password in  $c_0$ , and check it

⇒ perfect simulation of  $c_1$

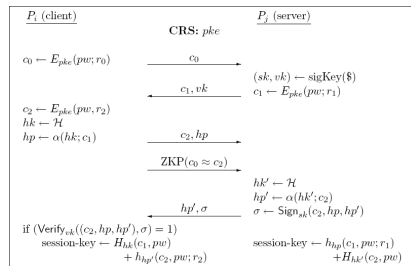
If  $\mathcal{A}$  plays the server role:

- $\mathcal{S}$  uses dummy password in  $c_0$
- From  $c_1$ ,  $\mathcal{S}$  extracts the committed password, and checks it

⇒ perfect simulation of  $c_2$  and ZKP

What about if  $\mathcal{A}$  corrupts the client right after  $c_0$ ?

⇒ security against static-corruptions only (before the session starts)



Non-malleable,  $L$ -extractable, equivocable commitment provides adaptive security to the KOY/GL construction

# Scheme II

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

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

- $\mathcal{S}$  can extract the committed password in  $c_0$ , and check it

⇒ perfect simulation of  $c_1$

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

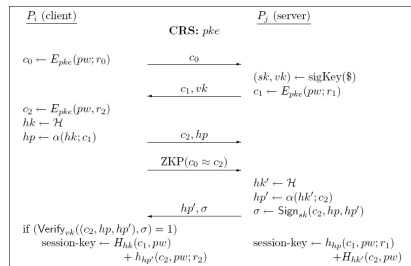
- $\mathcal{S}$  uses dummy password in  $c_0$
- From  $c_1$ ,  $\mathcal{S}$  extracts the committed password, and checks it

⇒ perfect simulation of  $c_2$  and ZKP

What about if  $\mathcal{A}$  corrupts the client right after  $c_0$ ?

⇒ security against static-corruptions only (before the session starts)

Non-malleable,  $L$ -extractable, equivocable commitment provides adaptive security to the KOY/GL construction



# Conclusion

## Smooth Projective Hash Functions for Complex Languages

Various Applications: in place of some ZK proofs

- first *efficient PAKE* with adaptive security in UC
- certification of keys (without costly ZK proofs of knowledge of sk)

# Conclusion

Smooth Projective Hash Functions for Complex Languages

Various Applications: in place of some ZK proofs

- first *efficient PAKE* with adaptive security in UC
- certification of keys (without costly ZK proofs of knowledge of sk)

# Conclusion

Smooth Projective Hash Functions for Complex Languages

Various Applications: in place of some ZK proofs

- first *efficient PAKE* with adaptive security in UC
- certification of keys (without costly ZK proofs of knowledge of sk)

# Conclusion

Smooth Projective Hash Functions for Complex Languages

Various Applications: in place of some ZK proofs

- first *efficient PAKE* with adaptive security in UC
- certification of keys (without costly ZK proofs of knowledge of sk)