Hash Proof Systems and Password Protocols

III – SPHF-based PAKE

David Pointcheval
CNRS, Ecole normale supérieure/PSL & INRIA

8th BIU Winter School – Key Exchange
February 2018

Intuition of PAKE with a Commitment

We denote $L_{pw}$ the language of the commitments of $pw$
- Alice sends $C_A$, a commitment of $pw_A$, to Bob (no leakage: hiding property)
- Bob can ask to verify that $C_A \in L_{pw_B}$:
  - Bob sends $hp_B$ to Alice, and computes $H_A \leftarrow \text{Hash}(hk_B, C_A)$
  - Alice can compute $pH_A \leftarrow \text{ProjHash}(hp, C_A, w_A)$
    \[ H_A = pH_A \iff pw_A = pw_B \]

Security: If $pw_B \neq pw_A$, $H_A$ is perfectly unpredictable to Alice (smoothness)

For a non-trivial language, the commitment must be perfectly binding
  e.g., ElGamal encryption: $C_A = (g^r, h^r \times g^{pw_A})$

SPHF-based PAKE: First Attempt

$X = \mathbb{G}^2$ and $L_{pw} = \{(g^r, h^r \times g^{pw})\}$
- Alice sends $C_A = (u = g^r, e = h^r \times g^{pw_A})$ to Bob
- Bob generates $hk = (\alpha, \beta) \xleftarrow{\$} \mathbb{Z}_p$ and sends $hp \leftarrow g^\alpha h^\beta$
- Bob computes $H \leftarrow u^\alpha(e/g^{pw_B})^\beta$
- Alice computes $pH \leftarrow hp^r$

\[ H_A = pH_A = g^{\alpha r} h^{\beta r} \iff pw_A = pw_B \]

Security: If $pw_B \neq pw_A$, $H$ is perfectly unpredictable to Alice (smoothness)

$C_A$ does not leak $pw_A$ under the DDH assumption
From the view of $pH$ (Reveal-query), Bob can look for $pw$ such that $u^\alpha(e/g^{pw})^\beta = pH$
\[ \implies \text{Off-line dictionary attack!} \]
SPHF-based PAKE

We denote $L_{pw}$ the language of the commitments of $pw$
- Alice sends $C_A$, a commitment of $pw_A$, to Bob (no leakage: hiding property)
- Bob can ask to verify that $C_A \in L_{pw_B}$:
  - Bob sends $hp_B$ to Alice, and computes $H_A \leftarrow \text{Hash}(hk_B, C_A)$
  - Alice can compute $pH_A \leftarrow \text{ProjHash}(hp, C_A, w_A)$
  
  $$H_A = pH_A \iff pw_A = pw_B$$

Bob must also prove his knowledge of $pw_B = pw_A$ before having access to $pH$
- Either with an implicit proof [Gennaro–Lindell – Eurocrypt ’03]
- Or with an explicit proof [Groce-Katz – CCS ’10]
SPHF-based PAKE: Implicit Proof

We denote \( L_A/L_B \) the languages of the commitments of \( pw_A/pw_B \)

- Alice sends \( C_A \), a commitment of \( pw_A \), to Bob
- Bob can ask to verify that \( C_A \in L_B \):
  - Bob sends \( hp_B \) to Alice, and computes \( H_A \leftarrow Hash_B(hk_B, C_A) \)
  - Alice can compute \( pH_A \leftarrow ProjHash_A(hp_B, C_A, w_A) \)
- Bob sends \( C_B \), a commitment of \( pw_B \), to Alice
- Alice can ask to verify that \( C_B \in L_A \):
  - Alice sends \( hp_A \) to Bob, and computes \( H_B \leftarrow Hash_A(hk_A, C_B) \)
  - Bob can compute \( pH_B \leftarrow ProjHash_B(hp_A, C_B, w_B) \)
- Bob computes \( K_B \leftarrow H_A \oplus pH_B \)
- Alice computes \( K_A \leftarrow pH_A \oplus H_B \)

\( K_B = H_A \oplus pH_B = pH_A \oplus H_B = K_A \iff pw_A = pw_B \)

SPHF-based PAKE: Man-In-The-Middle Attack

\( \mathcal{X} = \mathbb{G}^2 \) and \( L_{pw} = \{(g^r, h^r \times g^{pw})\} \)

- Alice sends \( C_A = (u_A = g^{\alpha_A}, e_A = h^{\beta_A} \times g^{pw_A}) \) to Bob
- Bob generates \( hk_B = (\alpha_B, \beta_B) \triangleleft \mathbb{Z}_p \) and sends \( hp_B \leftarrow g^{\alpha_B} h^{\beta_B} \)
- Bob sends \( C_B = (u_B = g^{\alpha_B}, e_B = h^{\beta_B} \times g^{pw_B}) \) to Bob
- Alice generates \( hk_A = (\alpha_A, \beta_A) \triangleleft \mathbb{Z}_p \) and sends \( hp_A \leftarrow g^{\alpha_A} h^{\beta_A} \)
- Alice computes \( K_A \leftarrow u_B^{\alpha_A} \cdot (e_B/g^{pw_A})^{\beta_A} \cdot hp_B^{\alpha_A} \)
- Bob computes \( K_B \leftarrow hp_B^{\alpha_A} \cdot u_B^{\beta_A} \cdot (e_A/g^{pw_B})^{\beta_B} \)

\( K_A = K_B \iff pw_A = pw_B \)

The adversary can do a man-in-the-middle attack:

- forwards everything
  - excepted \( C_B \) to Alice, that is replaced by \( C'_B = C_B \times (g, h) \)
  - \( K'_A = u_B^{\alpha_A} g^{\alpha_A} \cdot (e_B/g^{pw_A})^{\beta_A} h^{\beta_A} \cdot hp_B^{\alpha_A} = K_A \times g^{\alpha_A} h^{\beta_A} = K_B \times hp_A \)
SPHF-based PAKE: Man-In-The-Middle Attack

From the man-in-the-middle attack:
- the adversary can ask for a Reveal-query to Alice
- the adversary can ask for a Test-query to Bob (the session ID’s are different)
- the adversary can check the relation between the keys to decide on $b'$

The commitment $C_B$ must be non-malleable or confirmed to Bob

GL-PAKE

[GL-PAKE: Security Proof]

Send-queries to Bob: Oracle-Generated $C_A$ with $pw_A = pw_B = pw$

- Oracle-generated $C_A$ should imply oracle-generated $hp_A$
- Correctness
- Oracle-generated $hp_A$ should confirm $hp_B$: Correctness
- IND-CPA
GL-PAKE: Security Proof

Send-queries to Bob: Oracle-Generated $C_A$ with $pw_A \neq pw_B$

- Smoothness
- IND-CPA

Send-queries to Bob: Non Oracle-Generated $C_A$

- The adversary must encrypt the correct password: password-guessing probability
- Smoothness
- IND-CPA

Send-queries to Alice: Oracle-Generated $C_B$

Oracle-Generated $C_A$

Non Oracle-Generated $C_A$

Smoothness
GL-PAKE: Security Proof

To be more precise, in the final game

- The Execute-queries just work as Send-queries with oracle-generated flows
- The actual passwords are not set at the beginning, but randomly chosen at the end
- WIN = a random password (with Player ID) is in $P$: the probability is $q_S/N$

Encryption schemes:

- $$(\text{Enc}, \text{Dec})$$: SPHF-friendly L-IND-CCA encryption scheme $\ell = (A, B, vk)$
  $\implies$ where $vk$ is the verification key of a OT-Signature
  $\implies$ Labeled Cramer-Shoup Encryption
- $$(\text{Enc}', \text{Dec}')$$: SPHF-friendly IND-CPA encryption scheme
  $\implies$ ElGamal Encryption
- $$(C_A, hp_B, C_B, hp_A)$$ signed by $A$: OT-signature $(sk, vk)$
  $\implies$ an oracle-generated $C_A$ implies the same oracle-generated $hp_A$,
  and confirms the received $(hp_B, C_B)$
Cramer-Shoup Encryption Scheme is an L-IND-CCA PKE:
\[ C = (u_1 = g_1^r, u_2 = g_2^r, e = h^m, v = (cd^l)^{\gamma}) \text{ with } t = H(\ell, u_1, u_2, e) \]

C is a CS ciphertext of pw iff \((u_1, u_2, e/pw, v)\) is an \(r\)-th power of \((g_1, g_2, h, cd^l)\)

\[
\text{HashKG}() : h_k = (\alpha, \beta, \gamma, \delta) \xleftarrow{\$} \mathbb{Z}_q^4 \\
\text{ProjKG}(hk, C) : h_p = g_1^\alpha g_2^\beta h^\gamma (cd^l)^\delta \\
\text{Hash}(hk, C) : H = u_1^\alpha u_2^\beta (e/pw)^\gamma v^\delta \\
\text{ProjHash}(hp, C, r) : pH = hp^r
\]

This is not a CS-SPHF, hence the GL relaxation [Gennaro-Lindell – Eurocrypt '03]

---

**GL-PAKE: Complete Protocol** [Gennaro-Lindell – Eurocrypt '03]

Alice

\[(sk, vk) \xleftarrow{\$} \text{SignKG}(); \ell = (A, B, vk) \]

\[r_A \xleftarrow{\$} \mathbb{Z}_q; C_A \leftarrow \text{CS}'(pw_A, r_A)\]

\[ph_A \leftarrow \text{ProjHash}_A(hp_B, C_A)\]

\[hk_A \xleftarrow{\$} \text{HashKG}(); h_p \leftarrow \text{ProjKG}(hk_B, C_A)\]

\[H_A \rightarrow \text{Hash}_B(hk_A, C_B)\]

\[\Sigma \rightarrow \text{Sign}(sk, (C_A, hp_B, C_B, hp_A))\]

\[K_A \leftarrow H_A \times pH_A\]

Bob

\[h_B \xleftarrow{\$} \text{HashKG}(); hp_B \leftarrow \text{ProjKG}(hk_A, C_B)\]

\[h_B \xleftarrow{\$} \mathbb{Z}_q; C_B \leftarrow \text{EG}(pw_B, r_B)\]

\[hp_B, C_B \rightarrow \text{ProjHash}_B(hp_A, C_B, r_B)\]

Verif\((vk, (C_A, hp_B, C_B, hp_A), \Sigma)?\)

\[K_B \leftarrow H_A \times pH_B\]

A key confirmation can be added to the third flow: Explicit Authentication of Alice

---

**Outline**

- Introduction
- **Game-based Security**
  - Gennaro-Lindell PAKE
  - Groce-Katz PAKE
  - Improvements
- **Universal Composability**
  - UC-Secure PAKE: Static Corruptions
  - UC-Secure PAKE: Adaptive Corruptions
- Conclusion
We denote $L_A/L_B$ the languages of the commitments $C$ of $pw_A/pw_B$

- Alice sends $C_A$, a commitment of $pw_A$ with random coins $r_A$, to Bob
- Bob can ask to verify that $C_A \in L_B$:
  - Bob sends $hp_B$ to Alice, and computes $H_A \leftarrow \text{Hash}_B(hk_B, C_A)$
  - Alice can compute $pH_A \leftarrow \text{ProjHash}_A(hp, C_A, r_A)$
- Alice parses $pH_A$ as $r_B' \parallel K_A$
- Bob parse $H_A$ as $r_B \parallel K_B$

- Bob sends $C_B$, a commitment of $pw_B$ with random coins $r_B$, to Alice
- Alice can recompute the commitment $C'_B$ of $pw_A$ with random coins $r'_B$ and check whether $C'_B = C_B$

For a non-trivial language, the commitment $C_A$ must be perfectly binding
To avoid false positive on $C'_B = C_B$, the commitment $C_B$ must be perfectly binding e.g., Public-Key Encryption Scheme

---

**GK-PAKE: Security Proof**

Send-query to Alice: Oracle-Generated $C_B$ with $pw_B = pw_A = pw$

- $C_B$ must be specific to this execution
- Oracle-Generated $C_B$ must imply Oracle-Generated $hp_B$
- Correctness
Send-query to Alice: Oracle-Generated $C_B$ with $pw_B \neq pw_A$

- $pw_A \neq pw_B \implies C'_B \neq C_B$

Send-query to Alice: Non Oracle-Generated $C_B$

- $C_B$ specific to this execution, and non-malleable
- The adversary must encrypt the correct password: password-guessing probability
- $Dec'(C_B) \neq pw_A \implies C'_B \neq C_B$

Send-query to Alice: Oracle-Generated $C_B$

Send-query to Alice: Non Oracle-Generated $C_B$

IND-CPA
GK-PAKE: Security Proof

Send-query to Bob: Oracle-Generated $C_A$

- Correctness + IND-CCA

Send-query to Bob: Non Oracle-Generated $C_A$

- The adversary must encrypt the correct password: password-guessing probability
- Smoothness + IND-CCA

No abort anymore: difference if the guesses are correct
GK-PAKE: Security Proof

To be more precise, in the final game

- The Execute-queries just work as Send-queries with oracle-generated flows
- The actual passwords are not set at the beginning, but randomly chosen at the end
- \( WIN = \) a random password (with Player ID) is in \( \mathcal{P} \): the probability is \( q_S / N \)

Encryption schemes:

- \((\text{Enc}, \text{Dec})\): SPHF-friendly IND-CPA encryption scheme
  \( \Rightarrow \) ElGamal Encryption
- \((\text{Enc}', \text{Dec}')\): L-IND-CCA encryption scheme:
  \( \ell' = (A, B, C_A, hp_B) \)
  \( \Rightarrow \) this makes \( C_B \) specific to this execution because of \( C_A \)
  \( \Rightarrow \) an oracle-generated \( C_B \) implies the same oracle-generated \( hp_B \)
  \( \Rightarrow \) Labeled Cramer-Shoup Encryption

Alice generates and sends \( C_A = (u \leftarrow g^{r_A}, e \leftarrow h^{r_A} g^{pw_A}) \in G^2 \)

Bob

- generates \( h k_B = (\alpha, \beta) \) and \( h p_B = g^\alpha h^\beta \)
- computes \( r_B || K_B = KDF(u^\alpha, (e/g^{pw_A})^\beta) \)
- generates \( C_B = (u_1 = g_1^\alpha, u_2 = g_2^\beta, e = h^\alpha g^{pw_A}, v = (cdt)^\beta) \),
  with \( t = H(A, B, C_A, hp_B, u_1, u_2, e) \)
- sends \( (hp_B, C_B) \in G^5 \)

Alice

- computes \( r_B' || K_A = KDF(hp_B^\gamma) \)
- generates \( C'_A = (u'_1 = g_1^\gamma, u'_2 = g_2^\beta, e = h^\beta g^{pw_A}, v' = (cdt')^\gamma) \),
  with \( t' = H(A, B, C_A, hp_B, u'_1, u'_2, e') \)
- aborts if \( C'_B \neq C_B \)

A key confirmation can be added to the second flow: Explicit Authentication of Bob

Outline

- Introduction

1. **Game-based Security**
   - Gennaro-Lindell PAKE
   - Groce-Katz PAKE
   - Improvements

2. **Universal Composability**
   - UC-Secure PAKE: Static Corruptions
   - UC-Secure PAKE: Adaptive Corruptions

- Conclusion
**IND-PCA Encryption from KEM**

KeyGen() : $(sk, pk) \leftarrow EncapsKG()$
$hk \leftarrow HashKG(), hp \leftarrow ProjKG(hk)$
$sk' = (sk, hk), pk' = (pk, hp)$

Enc$(pk') = (pk, hp), m)$ : $(K, c) \leftarrow Encaps(pk, r), e \leftarrow m + K$
$\chi = (K, c), \nu = t$
$t = H(c, e), v = ProjHash(hp, \chi, t, \nu)$
$C = (c, e, v)$

Dec$(sk', C = (c, e, v))$ : $K \leftarrow Decaps(sk, c), m \leftarrow e - K$
$\chi = (K, c), t \leftarrow H(c, e), v' \leftarrow Hash(hk, \chi, t)$
If $(v' \neq v)$ $\Rightarrow$ Reject
Else Return $m$

**IND-PCA Security Proof**

- Smoothness (soundness): $v'' = v' \Rightarrow e' - m' \text{ valid key} \Rightarrow m' \text{ valid plaintext}$
- 2-Universal Smoothness (simulation-soundness)
- Correctness + Indistinguishability/Hard Subset Membership $\Rightarrow Pr[b' = b] = \frac{1}{2}$
GK-SPOKE (Simple Password-Only Key Exchange)  

[Abdalla-Benhamouda-P. – PKC ’15]

Alice generates and sends $C_A = (u \leftarrow g^A, e \leftarrow H^A g^{pwA}) \in \mathbb{G}^2$.

Bob

- generates $hk_B = (\alpha, \beta)$ and $hp_B = g^\alpha h^\beta$
- computes $r_B||K_A = KDF((u^A, (e/g^{pwA})^\beta))$
- generates $C_B = (u = g_1^\alpha, e = g_2^\beta g^{pwB}, v = (cdt)^\gamma),\quad t = \mathcal{H}(A, B, C_A, hp_B, u, e)$
- sends $(hp_B, C_B) \in \mathbb{G}^4$

Alice

- computes $r_B||K_A = KDF(hp_B^A)$
- generates $C_B = (u' = g_1^\alpha, e' = g_2^\beta g^{pwA}, v' = (cdt')^\gamma),\quad t' = \mathcal{H}(A, B, C_A, hp_B, u', e')$
- aborts if $C_B \neq C_B$

Instead of 7 group elements, only 6 group elements with a 2-flow protocol.

GL-PAKE: Reminder of the Idea

[Abbadino-Lindell – Eurocrypt ’03]

Alice

- Alice sends $C_A$, a commitment of $pw_A$, to Bob
- Bob can ask to verify that $C_A \in L_B$ (language of commitments of $pw_B$):
  - Bob sends $hp_B$ to Alice, and computes $H_A \leftarrow \text{Hash}_B(hk_B, C_A)$
  - Alice can compute $pH_A \leftarrow \text{ProjHash}_A(hp_B, C_A, w_A)$

Bob

- Bob sends $C_B$, a commitment of $pw_B$, to Alice
- Alice can ask to verify that $C_B \in L_A$ (language of commitments of $pw_A$):
  - Alice sends $hp_A$ to Bob, and computes $H_B \leftarrow \text{Hash}_A(hk_A, C_B)$
  - Bob can compute $pH_B \leftarrow \text{ProjHash}_B(hp_A, C_B, w_B)$

$$K_B = H_A \oplus pH_B = pH_A \oplus H_B = K_A \iff pw_A = pw_B$$
KV-PAKE: Katz-Vaikuntanathan’s Idea

Both are sent in parallel:
- Alice sends $C_A, h_{PA}$ to Bob
- Bob sends $C_B, h_{PB}$ to Alice

Upon reception of the partner’s flow:
- Alice computes $pH_A \leftarrow \text{ProjHash}_A(h_{PB}, C_A, w_A)$ and $H_B \leftarrow \text{Hash}_A(hk_A, C_B)$
- Bob computes $pH_B \leftarrow \text{ProjHash}_B(h_{PA}, C_B, w_B)$ and $H_A \leftarrow \text{Hash}_B(hk_B, C_A)$

Then

$$K_B = H_A \oplus pH_B \oplus pH_A \oplus H_B = K_A$$

KV Smoothness

[Abdalla-Benhamouda-P. – PKC ’15]

Alice generates

$$hk = (\alpha, \beta, \gamma, \delta) \text{ and } hp = (hp_1 \leftarrow g^\alpha \cdot h^\gamma \cdot c^\delta, hp_2 \leftarrow g^\beta \cdot d^\delta),$$

Hash$(hk,c) = u^{\alpha + \beta}(e/m)^{\gamma}v^{\delta} = (hp_1,hp_2)^r = \text{ProjHash}(hp,c,r)$

KV-SPOKE (Simple Password-Only Key Exchange)

[Abdalla-Benhamouda-P. – PKC ’15]

- Alice generates $hk_A = (\alpha, \beta, \gamma, \delta)$ and $hp_A = (hp_1 \leftarrow g^\alpha \cdot h^\gamma \cdot c^\delta, hp_2 \leftarrow g^\beta \cdot d^\delta)$
- $C_A = (u \leftarrow g^\alpha, e \leftarrow h^\gamma \cdot g^{ow_A}, v \leftarrow (cd^\delta)^\gamma)$ for $t_A = H(A, B, u, e, hp_A)$
- Alice sends $C_A \in \mathbb{G}^3$ and $hp_A \in \mathbb{G}^2$
- Alice receives $C_B = (u', e', v') \in \mathbb{G}^3$ and $hp_B = (hp_1', hp_2') \in \mathbb{G}^2$ from Bob
- Alice computes

$$t_B = H(B, A, u', e', hp_B)$$

$$H_B = u'^{\alpha + t_B}(e'/pw_A)^{\gamma}v^{\delta}$$

$$pH_A = (hp_1', hp_2')^{t_B}$$

$$K_A = pH_A \times H_B$$

Only 5 group elements sent by each player in a 2-simultaneously flow protocol
**Outline**

- **Introduction**
- **1 Game-based Security**
  - Gennaro-Lindell PAKE
  - Groce-Katz PAKE
  - Improvements
- **2 Universal Composability**
  - UC-Secure PAKE: Static Corruptions
  - UC-Secure PAKE: Adaptive Corruptions
- **Conclusion**

---

**SPHF-Based PAKE: Protocol**

**Alice**

- $hk_A \overset{\$}{\leftarrow} \text{HashKG}()$
- $hp_A \leftarrow \text{ProjKG}(hk_A)$
- $\ell_A = (A, B, hp_A), r_A \overset{\$}{\leftarrow} $, $C_A \leftarrow \text{Enc}^{\ell_A}(pw_A, r_A)$
- $H_B \leftarrow \text{Hash}_A(hk_A, C_B)$
- $pH_A \leftarrow \text{ProjHash}_A(hp_B, C_A, r_A)$
- $K_A \leftarrow H_B \times pH_A$

**Bob**

- $hk_B \overset{\$}{\leftarrow} \text{HashKG}()$
- $hp_B \leftarrow \text{ProjKG}(hk_B)$
- $\ell_B = (B, A, hp_B), r_B \overset{\$}{\leftarrow} $, $C_B \leftarrow \text{Enc}^{\ell_B}(pw_B, r_B)$
- $H_A \leftarrow \text{Hash}_B(hk_B, C_A)$
- $pH_B \leftarrow \text{ProjHash}_B(hp_A, C_B, r_B)$
- $K_B \leftarrow H_A \times pH_B$

UC-secure against static corruptions
SPHF-Based PAKE: Simulation

NewSession: for $U$ with $U'$
- $hk \xleftarrow{\$} \text{HashKG}(); hp \leftarrow \text{ProjKG}(hk)$
- $\ell = (U, U', hp), r \xleftarrow{\$} $, $C \leftarrow \text{Enc}^\ell(pw, r)$

Flow $hp', C'$
- Oracle-Generated from $U'$: $hk', hp' \leftarrow \text{ProjKG}(hk'), C' \leftarrow \text{Enc}^\ell(pw', r')$
  - $H \leftarrow \text{Hash}(hk', C')$
  - $\text{pH} = \text{ProjHash}(hp'(C', r'))$
  - $K \leftarrow H \times \text{pH}$

Passwords known for corrupted players

- Non Oracle-Generated: $pw' \leftarrow \text{Dec}^\ell(C')$ and $\text{TestPwd}(pw')$
  - $H = \text{Hash}(C', C, r)$
  - $\text{pH} \leftarrow \text{ProjHash}(hp', C, r)$
  - $K/\text{pH} / H/\text{pH} K \xleftarrow{\$} $ if incorrect guess

SPHF-Based PAKE: Simulation

NewSession: for $U$ with $U'$
- $hk \xleftarrow{\$} \text{HashKG}(); hp \leftarrow \text{ProjKG}(hk)$
- $\ell = (U, U', hp), r \xleftarrow{\$} $, $C \leftarrow \text{Enc}^\ell(pw, r)$

Flow $hp', C'$
- Oracle-Generated from $U'$: $hk', hp' \leftarrow \text{ProjKG}(hk'), C' \leftarrow \text{Enc}^\ell(pw', r')$
  - $H \leftarrow \text{Hash}(hk', C')$
  - $\text{pH} = \text{ProjHash}(hp'(C', r'))$
  - $K \leftarrow H \times \text{pH}$

Passwords known for corrupted players
**SPHF-Based PAKE: Protocol**

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_{k_A} \leftarrow \text{HashKG}(); h_{p_A} \leftarrow \text{ProjKG}(h_{k_A}) )</td>
<td>( h_{k_B} \leftarrow \text{HashKG}(); h_{p_B} \leftarrow \text{ProjKG}(h_{k_B}) )</td>
</tr>
<tr>
<td>( \ell_A = (A, B, h_{p_A}), r_A \leftarrow $, C_A \leftarrow \text{Com}^\ell_A(p_{w_A}, r_A) )</td>
<td>( \ell_B = (B, A, h_{p_B}), r_B \leftarrow $, C_B \leftarrow \text{Com}^\ell_B(p_{w_B}, r_B) )</td>
</tr>
<tr>
<td>( H_A \leftarrow \text{Hash}<em>A(h</em>{k_A}, C_B) )</td>
<td>( H_A \leftarrow \text{Hash}<em>A(h</em>{k_B}, C_A) )</td>
</tr>
<tr>
<td>( p_{H_A} \leftarrow \text{ProjHash}<em>A(h</em>{p_B}, C_A, r_A) )</td>
<td>( p_{H_B} \leftarrow \text{ProjHash}<em>B(h</em>{p_A}, C_B, r_B) )</td>
</tr>
<tr>
<td>( K_A \leftarrow H_B \times p_{H_A} )</td>
<td>( K_B \leftarrow H_A \times p_{H_B} )</td>
</tr>
</tbody>
</table>

With \( \text{Com} = \text{Enc} \), if \( r \) required for \( \text{ProjHash}(hp, C, r) \):

\[ \implies \text{no security against adaptive corruptions} \]

Extractable and equivocable commitment (i.e. UC-secure) and SPHF-friendly:

\[ \implies \text{security against adaptive corruptions} \]

**SPHF-Friendly Commitments**

- Based on Cramer-Shoup (extractability) and Pedersen (equivocability)
  Inspired from the Canetti-Fischlin commitment
  \[ \implies \text{Commitment size linear in } mk^2 \]  
  \[ \text{[Canetti-Fischlin – Crypto ‘01]} \]
  \[ \text{[Abdalla-Chevalier-P. – Crypto ‘09]} \]

- Improvement with Haralambiev (equivocability)
  \[ \implies \text{Commitment size linear in } mk \]  
  \[ \text{[Haralambiev – PhD Thesis ‘11]} \]
  \[ \text{[Abdalla-Benhamouda-Blazy-Chevalier-P. – Asiacrypt ‘13]} \]

- SPHF-Friendly variant of FLM commitment
  \[ \implies \text{Commitment size linear in } k \]  
  \[ \text{[Fischlin-Libert-Manulis – Asiacrypt ‘11]} \]
  \[ \text{[Blazy-Chevalier – Asiacrypt ‘16]} \]

\( m = \text{length of the password} \quad k = \text{security parameter} \)
Conclusion

In the line of the KOY protocol \[\text{Katz-Ostrovsky-Yung – Crypto '01}\] the GL methodology widely used for PAKE \[\text{Gennaro–Lindell – Eurocrypt '03}\]

- BPR-secure protocols
- \textbf{UC-secure} protocols for static corruptions \[\text{Canetti-Halevi-Katz-Lindell-MacKenzie – Eurocrypt '05}\]
- \textbf{UC-secure} protocols for \textit{adaptive corruptions} \[\text{Abdalla-Chevalier-P. – Crypto 09}\]
- \textbf{One-Round} protocols (BPR and UC) \[\text{Katz-Vaikuntanathan – TCC '11}\]
- \textbf{UC-secure} protocols for adaptive corruptions \textit{without erasures} \[\text{Abdalla-Benhamouda-P. – PKC '17}\]

Equivalently, SPHF can be used for Oblivious Transfer