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 $hk = (\alpha, \beta) \overset{\$}{\leftarrow} \mathbb{Z}_p$ and sends $hp \leftarrow g^{\alpha} h^{\beta}$
- Bob computes $H \leftarrow u^{\alpha} (e / g^{pw_B})^{\beta}$
  
  $$H_A = pH_A = g^{\alpha r} h^{\beta r} \iff pw_A = pw_B$$

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 $u^{\alpha} (e / g^{pw})^{\beta} = pH$

$\implies$ 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 $pH_A$

- 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 $H_A \leftarrow \text{ProjHash}_B(hp_B, C_A, w_A)$
  - Alice can compute $pH_A \leftarrow \text{ProjHash}_A(hp_B, C_A, w_A)$
- 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 $H_B \leftarrow \text{ProjHash}_A(hp_A, C_B, w_B)$
  - Bob can compute $pH_B \leftarrow \text{ProjHash}_B(hp_A, C_B, w_B)$
- Bob computes $K_B \leftarrow H_A \oplus pH_B$
- Alice computes $K_A \leftarrow pH_A \oplus H_B$

$K_A = K_B \iff pw_A = pw_B$

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

$X = G_2$ and $L_{pw} = \{(g^r, h^r \times g^{pw})\}$

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

$K_A = K_B \iff pw_A = pw_B$

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

- forwards everything
- excepted $C_B$ to Alice, that is replaced by $C'_B = C_B \times (g, h)$

$K'_A = u_B^{\alpha_A} g^{\alpha_A} \cdot (e_B/g^{pw_A})^{\beta_A} h^{\beta_A} \cdot hp_B^{\alpha_A} = K_A \times g^{\alpha_A h^{\beta_A}} = K_B \times hp_A$
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**

**[Gennaro-Lindell – Eurocrypt ’03]**

Alice

\[
\begin{align*}
    r_A &\xlongleftarrow{\$}; C_A \leftarrow \text{Enc}(pw_A, r_A) \\
    pH_A &\leftarrow \text{ProjHash}(h_B, C_A, r_A) \\
    h_B &\leftarrow \text{Hash}(h_B, C_A) \\
     K_A &\leftarrow H_B \times pH_A
\end{align*}
\]

Bob

\[
\begin{align*}
    h_B &\xlongleftarrow{\$}; \text{HashKG}(); hp_B \leftarrow \text{ProjKG}(h_B) \\
    H_A &\leftarrow \text{Hash}(h_B, C_A) \\
    C_B &\leftarrow \text{Enc}'(pw_B, r_B) \\
    pH_B &\leftarrow \text{ProjKG}(hp_B, C_B) \\
    K_B &\leftarrow pH_B \times H_A
\end{align*}
\]

Which are the security properties of the encryption schemes?

---

**GL-PAKE: Security Proof**

Send-queries to Bob: Oracle-Generated \( C_A \) with \( pw_A = pw_B = pw \)

Alice

\[
\begin{align*}
    r_A &\xlongleftarrow{\$}; C_A \leftarrow \text{Enc}(pw_A, r_A) \\
    pH_A &\leftarrow \text{ProjHash}(hp_B, C_A, r_A) \\
    h_B &\leftarrow \text{Hash}(h_B, C_A) \\
     K_A &\leftarrow H_B \times pH_A
\end{align*}
\]

Bob

\[
\begin{align*}
    h_B &\xlongleftarrow{\$}; \text{HashKG}(); hp_B \leftarrow \text{ProjKG}(h_B) \\
    H_A &\leftarrow \text{Hash}(h_B, C_A) \\
    C_B &\leftarrow \text{Enc}'(pw_B, r_B) \\
    pH_B &\leftarrow \text{ProjKG}(hp_B, C_B, r_B) \\
    pH_B &\leftarrow \text{ProjHash}(hp_A, C_B, r_B) \\
    K_B &\leftarrow K_A
\end{align*}
\]

- 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 \( C_B \)
GL-PAKE: Security Proof

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

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

GL-PAKE: Security Proof

Oracle-Generated $C_B$

Algorithm:

- $C_A \leftarrow \text{Enc}(\$, S)$
- $\ell \leftarrow \text{Enc}(\$, S)$
- $h_{k_A} \leftarrow \text{HashKG}(\ell)$
- $h_{k_A} \leftarrow \text{ProjKG}(\ell)$
- $K_A \leftarrow \$'

Oracle-Generated $C_A$

Bob

- $h_{k_B} \leftarrow \text{HashKG}(\ell)$
- $h_{k_B} \leftarrow \text{ProjKG}(\ell)$
- If $p_{\ell,A} = p_{\ell,B}$ and compatible oracles, $K_B \leftarrow K_A$
- If incompatible oracles, $K_B \leftarrow \$

Non Oracle-Generated $C_B$

Algorithm:

- $C_A \leftarrow \text{Enc}(\$, S)$
- $\ell \leftarrow \text{Enc}(\$, S)$
- $h_{k_A} \leftarrow \text{HashKG}(\ell)$
- $h_{k_A} \leftarrow \text{ProjKG}(\ell)$
- Dec($C_B$) in $P$

Non Oracle-Generated $C_A$

Bob

- Dec($C_A$) in $P$
- If $p_{\ell,A} = p_{\ell,B}$ and compatible oracles, $K_B \leftarrow K_A$
- If incompatible oracles, $K_B \leftarrow \$

IND-CCA + No abort anymore: difference if the guesses are correct

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 $\ell = (A, B, vk)$
  - where $vk$ is the verification key of a OT-Signature
  - $\Rightarrow$ Labeled Cramer-Shoup Encryption
- $(\text{Enc}', \text{Dec}')$: SPHF-friendly IND-CPA encryption scheme
  - $\Rightarrow$ ElGamal Encryption
- $(C_A, h_{p_B}, C_B, h_{p_A})$ signed by $A$: OT-signature $(sk, vk)$
  - an oracle-generated $C_A$ implies the same oracle-generated $h_{p_A}$, and confirms the received $(h_{p_B}, C_B)$
Cramer-Shoup Encryption Scheme is an L-IND-CCA PKE:

\[ C = (u_1 = g_1^r, u_2 = g_2^r, e = h^r m, v = (cd^i)^v) \] 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 \( (g_1, g_2, h, cd^i) \)

This is not a CS-SPHF, hence the GL relaxation

### GL-PAKE: Complete Protocol

**Alice**

\((sk, vk) \overset{\$}{\leftarrow} \text{SignKG}(); \ell = (A, B, vk)\)

\( r_A \overset{\$}{\leftarrow} \$; C_A \leftarrow \text{CS}'(pw_A, r_A) \)

\( pH_A \leftarrow \text{ProjHash}_A(hp_B, C_A) \)

\( hk_A \overset{\$}{\leftarrow} \text{HashKG}(); hp_A \leftarrow \text{ProjKG}(hk, C) \)

\( H_A \leftarrow \text{Hash}(hk, C) = u_1^r u_2^r (e/pw)^r v^r \)

\( \Sigma \leftarrow \text{Sign}(sk, (C_A, hp_B, C_B, hp_A)) \)

\( K_A \leftarrow H_A \times pH_A \)

**Bob**

\( h_B \overset{\$}{\leftarrow} \text{HashKG}(); hp_B \leftarrow \text{ProjKG}(hk_B, C_A) \)

\( H_B \leftarrow \text{Hash}(hk_B, C_A) \)

\( r_B \overset{\$}{\leftarrow} \$; C_B \leftarrow \text{EG}(pw_B, r_B) \)

\( pH_B \leftarrow \text{ProjHash}_B(hp_A, C_B, r_B) \)

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

\( K_B \leftarrow H_A \times pH_B \)

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**
SPHF-based PAKE: Explicit Proof

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 $r_B' || K_A$
- Bob parses $H_A$ as $r_B || K_B$
- 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

[Groce-Katz – CCS ‘10]

Which are the security properties of the encryption schemes?

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
GK-PAKE: Security Proof

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
- $\text{Dec}'(C_B) \neq pw_A \implies C'_B \neq C_B$

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

Oracle-Generated $C_B$

- No abort anymore: difference if the guesses are correct

Non Oracle-Generated $C_B$

- Dec($C_B$) in $P$
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
  $\implies$ ElGamal Encryption
- $(\text{Enc}', \text{Dec}')$: L-IND-CCA encryption scheme: $t' = (A, B, C_A, h_{PB})$
  $\implies$ this makes $C_B$ specific to this execution because of $C_A$
  $\implies$ an oracle-generated $C_B$ implies the same oracle-generated $h_{PB}$
  $\implies$ Labeled Cramer-Shoup Encryption

GK-PAKE

[Groce-Katz – CCS ’10]

- Alice generates and sends $C_A = (u \leftarrow g^{r_A}, e \leftarrow h_A^{r_A} g_{PW_A}) \in \mathbb{G}^2$
- Bob
  - generates $h_{kB} = (\alpha, \beta)$ and $h_{PB} = g^{\alpha} h^{\beta}$
  - computes $r_B || K_B = KDF(u^\alpha, (e/g_{PW_B})^\beta)$
  - generates $C_B = (u_1 = g_1^{e_1}, u_2 = g_2^{e_2}, e = h^{e_2} g_{PW_B}, v = (cdt)^{r_B})$
    with $t = H(A, B, C_A, h_{PB}, u_1, u_2, e)$
  - sends $(h_{PB}, C_B) \in \mathbb{G}^5$
- Alice
  - computes $r_B' || K_A = KDF(h_{PB}'^\beta)$
  - generates $C_A' = (u_1 = g_1^{e_1}, u_2 = g_2^{e_2}, e = h^{e_2} g_{PW_A}, v' = (cdt')^{r_B'})$
    with $t' = H(A, B, C_A, h_{PB}, u_1, u_2, e')$
  - aborts if $C_A' \neq C_B$

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

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
Better Efficiency

IND-PCA

Security proofs: WIN = a random password (with Player ID) is in \( P \)

- One either decrypts every \( C \) into \( pw \) and checks whether \( pw \in P \) or not
  \( \implies \) need of decryption oracle
- Or one checks for every \( pw \in P \) whether \( C \) encrypts \( pw \) for some ciphertext \( C \)
  \( \implies \) need of plaintext-checking oracle

The previous proofs work for IND-PCA encryption schemes, instead of IND-CCA

KV-SPHF

Number of flows in GL-PAKE: 3 flows because \( hp \) depends on/after \( x \)

With \( hp \) possibly sent before \( x \): 1-round protocol with KV-SPHF

IND-PCA Encryption from KEM

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

Enc(\( pk' = (pk, hp) \), \( m \)) : \(( K, c) \leftarrow Encaps(pk, r), e \leftarrow m + K \)
\( x' = (K, c), w = r \)
\( t = \mathcal{H}(c, e), v = ProjHash(hp, x', t, w) \)
\( C = (c, e, v) \)

Dec(\( sk' = (sk, hk) \), \( C = (c, e, v) \)) : \( K \leftarrow Decaps(sk, c) \), \( m \leftarrow e - K \)
\( x' = (K, c), t \leftarrow \mathcal{H}(c, e), v' \leftarrow Hash(hk, x', t) \)
If \( v' \neq v \) \( \implies \) Reject
Else Return \( m \)

IND-PCA Security Proof

\[ (sk, pk) \leftarrow EncapsKG(), hk \leftarrow HashKG(), hp \leftarrow ProjKG(hk) \]
\[ b \leftarrow \{0, 1\} \]
\[ k' \leftarrow Encaps(pk, u) \]
\[ x' = (sk, k'), (K, c) \leftarrow Encaps(pk, c) \]
\[ t = \mathcal{H}(c, e), v = ProjHash(hp, x', t, w) \]
\[ C = (c, e, v) \]
\[ \begin{align*}
  b' \leftarrow & b \\
  C' \leftarrow & (c', e', v') \\
  m' \leftarrow & (m') \\
  k' \leftarrow & (sk, k') \\
  m' \leftarrow & \mathcal{H}(c', e') 
\end{align*} \]

PCA(\( sk', m', C' \))

- \( k' \leftarrow Encaps(sk', c') \)
- \( m' \leftarrow Encaps(pk, u) \)
- \( v' \leftarrow Hash(hk, x', t') \)
- \( v'' \leftarrow Hash(hk, x', t') \)
- \( b' \leftarrow b \)

Smootheness (soundness): \( v'' = v' \implies e' - m' \) valid key \( \implies m' \) valid plaintext
+ 2-Universal Smoothness
Correctness + Indistinguishability/Hard Subset Membership \( \implies \Pr[b' = b] = \frac{1}{2} \)
**GL-PAKE: Reminder of the Idea**

**[Gennaro-Lindell – Eurocrypt ’03]**

**Alice**
- Alice sends $C_A = (u \leftarrow g^A, e \leftarrow h^A g^{\text{pw}_A}) \in \mathbb{G}^2$
- Bob
  - generates $h_k_B = (\alpha, \beta)$ and $h_{\text{pw}B} = g^\alpha h^\beta$
  - computes $r_B \| K_B = KDF(u^\alpha \cdot (e / g^{\text{pw}_B})^\beta)$
  - generates $C_B = (u = g_B^t, e = g_B^e g^{\text{pw}_B}, v = (cdt)^e)$,
    with $t = H(A, B, C_A, h_{\text{pw}B}, u, e)$
  - sends $(h_{\text{pw}B}, C_B) \in \mathbb{G}^4$
- Alice
  - computes $r_B' \| K_A = KDF(h_B^g)$
  - generates $C_B' = (u' = g_A^t, e' = g_A^e g^{\text{pw}_B}, v' = (cdt')^e)$,
    with $t' = H(A, B, C_A, h_{\text{pw}B}, u', e')$
  - aborts if $C_B' \neq C_B$

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

**Bob**
- Bob sends $C_B$, a commitment of $\text{pw}_B$, to Alice
- Alice can ask to verify that $C_B \in \mathcal{L}_B$ (language of commitments of $\text{pw}_B$):
  - Bob sends $h_{\text{pw}B}$ to Alice, and computes $H_A \leftarrow \text{Hash}_B(h_k_B, C_A)$
  - Alice can compute $pH_A \leftarrow \text{ProjHash}_A(h_{\text{pw}B}, C_A, z_A)$
- Bob sends $C_B$, a commitment of $\text{pw}_B$, to Alice
- Alice can ask to verify that $C_B \in \mathcal{L}_A$ (language of commitments of $\text{pw}_A$):
  - Alice sends $h_{\text{pw}A}$ to Bob, and computes $H_B \leftarrow \text{Hash}_A(h_k_A, C_B)$
  - Bob can compute $pH_B \leftarrow \text{ProjHash}_B(h_{\text{pw}A}, C_B, z_B)$

$$K_B = H_A \oplus pH_B = pH_A \oplus H_B = K_A \iff \text{pw}_A = \text{pw}_B$$
KV-PAKE: Katz-Vaikuntanathan’s Idea

Both are sent in parallel:

- Alice sends $C_A, hp_A$ to Bob
- Bob sends $C_B, hp_B$ to Alice

Upon reception of the partner’s flow:

- Alice computes $pH_A \leftarrow \text{ProjHash}_A(hp_B, C_A, w_A)$ and $H_B \leftarrow \text{Hash}_A(hk_A, C_B)$
- Bob computes $pH_B \leftarrow \text{ProjHash}_B(hp_A, C_B, w_B)$ and $H_A \leftarrow \text{Hash}_B(hk_B, C_A)$

Then

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

KV Smoothness

$\forall f \rightarrow X \setminus \mathcal{L}$, with the probability space $hk \overset{\$}{\leftarrow} \text{HashKG}(), hp \leftarrow \text{ProjKG}(hk)$

$\{(hp, H) \mid H \leftarrow \text{Hash}(hk, f(hp))\} \approx \{(hp, H) \mid H \overset{\$}{\leftarrow} \Pi\}$

Given $hp$, no adversary can find $x \in X \setminus \mathcal{L}$ for which it can distinguish $\text{Hash}(hk, x)$

KV-SPHF for SCS Ciphertexts of $m$

$\quad c = (u = g^t, e = h^e m, v = (cd^t)^\gamma)$ with $t = \mathcal{H}(u, e)$

$\quad hk = (\alpha, \beta, \gamma, \delta)$

$\quad hp = (hp_1 \leftarrow g^\alpha h^\gamma c^\delta, hp_2 \leftarrow g^\beta d^\delta)$

$\quad \text{Hash}(hk, c) = u^{t^{} + \gamma \beta} (e/m)^\gamma v^\delta = (hp_1 h_2^\beta)^\gamma = \text{ProjHash}(hp, c, r)$

KV-SPOKE (Simple Password-Only Key Exchange)

- Alice generates
  - $hk_A = (\alpha, \beta, \gamma, \delta)$ and $hp_A = (hp_1 \leftarrow g^\alpha h^\gamma c^\delta, hp_2 \leftarrow g^\beta d^\delta)$
  - $C_A = (u \leftarrow g^\delta, e \leftarrow h^e g^{ow_A}, v \leftarrow (cd^w_A)^\alpha)$ for $t_A = \mathcal{H}(A, B, u, e, hp_A)$
- 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
  - $t_B = \mathcal{H}(B, A, u', e', hp_B)$
  - $H_B = u'^{\alpha + bh^\beta} (e'/pw_A)^\gamma v^\delta$
  - $pH_A = (hp_1 hp_2' u')^\alpha$
  - $K_A = pH_A \times H_B$

Only 5 group elements sent by each player in a 2-simultaneously flow protocol
SPHF-Based PAKE: Protocol

Alice

\[ hk_A \leftarrow \text{HashKG}(); \quad hp_A \leftarrow \text{ProjKG}(hk_A) \]
\[ \ell_A = (A, B, hp_A), \quad r_A \leftarrow \$; \quad C_A \leftarrow \text{Enc}_\ell^{\ell}(pw_A, r_A) \]
\[ H_B \leftarrow \text{Hash}_A(hk_A, C_B) \]
\[ pH_A \leftarrow \text{ProjHash}_A(hp_B, C_A, r_A) \]
\[ K_A \leftarrow H_B \times pH_A \]

Bob

\[ hk_B \leftarrow \text{HashKG}(); \quad hp_B \leftarrow \text{ProjKG}(hk_B) \]
\[ \ell_B = (B, A, hp_B), \quad r_B \leftarrow \$; \quad C_B \leftarrow \text{Enc}_\ell^{\ell}(pw_B, r_B) \]
\[ H_A \leftarrow \text{Hash}_B(hk_B, C_A) \]
\[ pH_B \leftarrow \text{ProjHash}_B(hp_A, C_B, r_B) \]
\[ K_B \leftarrow H_A \times pH_B \]

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

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

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

- **Non Oracle-Generated:** $pw' \leftarrow \text{Dec}^\ell(C')$ and $\text{TestPwd}(pw')$
  - $H \leftarrow \text{Hash}(hk', C')$
  - $pH \leftarrow \text{ProjKG}(hp', C, r, pH)$
  - $K \leftarrow H \times pH$

**Passwords known for corrupted players**

**SPHF-Based PAKE: Simulation**

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

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

- **Non Oracle-Generated:** $pw' \leftarrow \text{Dec}^\ell(C')$ and $\text{TestPwd}(pw')$
  - $H \leftarrow \text{Hash}(hk', C')$
  - $pH \leftarrow \text{ProjKG}(hp', C, r, pH)$
  - $K \leftarrow H \times pH$

**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}() ) ( \rightarrow h_p_A \leftarrow \text{ProjKG}(h_k_A) )</td>
<td>( h_k_B \leftarrow \text{HashKG}() ) ( \rightarrow h_p_B \leftarrow \text{ProjKG}(h_k_B) )</td>
</tr>
<tr>
<td>( \ell_A = (A, B, h_p_A), r_A \leftarrow $ ) ( \rightarrow C_A \leftarrow \text{Com}^A(p_w_A, r_A) )</td>
<td>( \ell_B = (B, A, h_p_B), r_B \leftarrow $ ) ( \rightarrow C_B \leftarrow \text{Com}^B(p_w_B, r_B) )</td>
</tr>
<tr>
<td>( H_B \leftarrow \text{Hash}_A(h_k_A, C_B) )</td>
<td>( H_B \leftarrow \text{Hash}_B(h_k_B, C_A) )</td>
</tr>
<tr>
<td>( p_H_A \leftarrow \text{ProjHash}_A(h_p_B, C_A, r_A) )</td>
<td>( p_H_B \leftarrow \text{ProjHash}_B(h_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 \( \text{ProjHash}(h_p, C, r) \):

- \( \Rightarrow \) no security against **adaptive corruptions**
- Extractable and equivocable commitment (i.e. UC-secure) and SPHF-friendly:
  - \( \Rightarrow \) security against **adaptive corruptions**

---

**SPHF-Friendly Commitments**

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

- Improvement with Haralambiev (equivocability)
  \( \Rightarrow \) Commitment size linear in \( mk \)

- SPHF-Friendly variant of FLM commitment
  \( \Rightarrow \) Commitment size linear in \( k \)
  [Fischlin-Libert-Manulis – Asiacrypt ’11] [Blazy-Chevalier – Asiacrypt ’16]

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

Conclusion

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

- BPR-secure protocols
- UC-secure protocols for adaptive corruptions [Abdalla-Chevalier-P. – Crypto 09]
- One-Round protocols (BPR and UC) [Katz-Vaikuntanathan – TCC ’11]
- UC-secure protocols for adaptive corruptions without erasures [Abdalla-Benhamouda-P. – PKC ’17]

Equivalently, SPHF can be used for Oblivious Transfer