

III – SPHF-based PAKE

David Pointcheval

CNRS, Ecole normale supérieure/PSL & INRIA



8th BIU Winter School – Key Exchange
February 2018

CNRS/ENS/PSL/INRIA

David Pointcheval

1/53

Intuition of PAKE with a Commitment

We denote \mathcal{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 $C_A \in \mathcal{L}_{pw_B}$:
 - 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: $C_A = (g^r, h^r \times g^{pw_A})$

CNRS/ENS/PSL/INRIA

David Pointcheval

2/53

SPHF-based PAKE: First Attempt

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

- 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 $pH \leftarrow hp^r$
- $$\left. \begin{array}{l} \text{Bob computes } H \leftarrow u^\alpha (e/g^{pw_B})^\beta \\ \text{Alice computes } pH \leftarrow hp^r \end{array} \right\} H_A = pH_A = g^{\alpha r} h^{\beta r} \iff pw_A = pw_B$$

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!**

CNRS/ENS/PSL/INRIA

David Pointcheval

3/53

SPHF-based PAKE

We denote \mathcal{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 $C_A \in \mathcal{L}_{pw_B}$:
 - 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]

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

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

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: Implicit Proof

We denote $\mathcal{L}_A/\mathcal{L}_B$ 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 $C_A \in \mathcal{L}_B$:
 - Bob sends hp_B to Alice, and computes $H_A \leftarrow \text{Hash}_B(hk_B, C_A)$
 - Alice can compute $pH_A \leftarrow \text{ProjHash}_A(hp_B, C_A, w_A)$
- Bob sends C_B , a commitment of pw_B , to Alice
- Alice can ask to verify that $C_B \in \mathcal{L}_A$:
 - Alice sends hp_A to Bob, and computes $H_B \leftarrow \text{Hash}_A(hk_A, C_B)$
 - Bob can compute $pH_B \leftarrow \text{ProjHash}_B(hp_A, C_B, w_B)$
- 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 $\mathcal{L}_{pw} = \{(g^r, h^r \times g^{pw})\}$

- Alice sends $C_A = (u_A = g^{r_A}, e_A = h^{r_A} \times g^{pw_A})$ 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 $C_B = (u_B = g^{r_B}, e_B = h^{r_B} \times g^{pw_B})$ 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 $K_A \leftarrow u_B^{\alpha_A} \cdot (e_B/g^{pw_A})^{\beta_A} \times hp_B^{r_A}$
 - Bob computes $K_B \leftarrow hp_A^{r_B} \times u_A^{\alpha_B} \cdot (e_A/g^{pw_B})^{\beta_B}$
- $$\left. \begin{array}{l} \text{Alice computes } K_A \leftarrow u_B^{\alpha_A} \cdot (e_B/g^{pw_A})^{\beta_A} \times hp_B^{r_A} \\ \text{Bob computes } K_B \leftarrow hp_A^{r_B} \times u_A^{\alpha_B} \cdot (e_A/g^{pw_B})^{\beta_B} \end{array} \right\} K_A = K_B \iff pw_A = pw_B$$

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

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

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

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

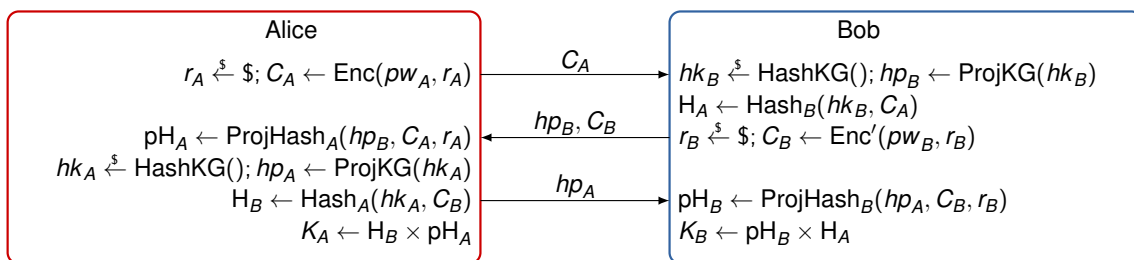
From the man-in-the-middle attack:

- 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

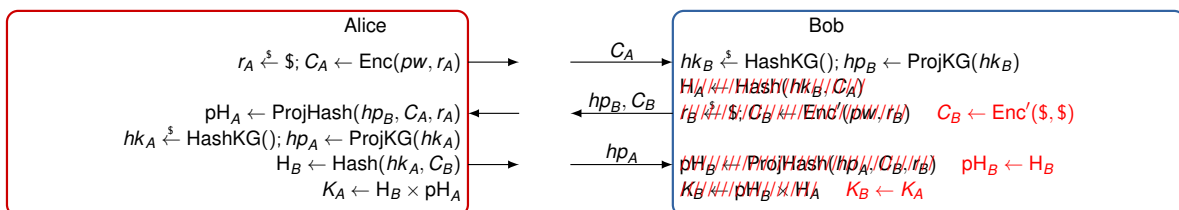
[Gennaro-Lindell – Eurocrypt '03]



Which are the security properties of the encryption schemes ?

GL-PAKE: Security Proof

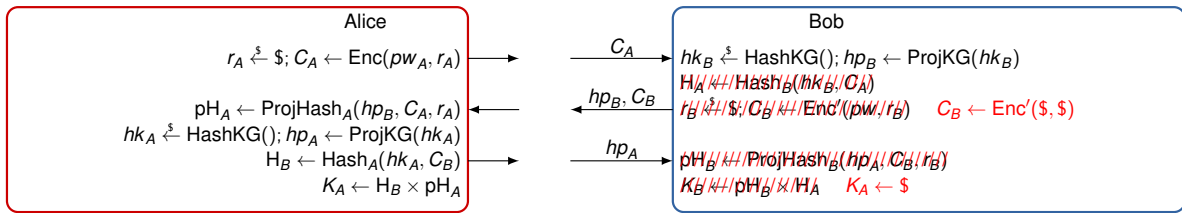
Send-queries to Bob: Oracle-Generated C_A with $pw_A = pw_B = pw$



- 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

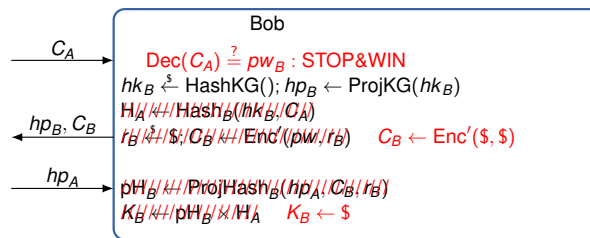
Send-queries to Bob: Oracle-Generated C_A with $pw_A \neq pw_B$



- Smoothness
- IND-CPA

GL-PAKE: Security Proof

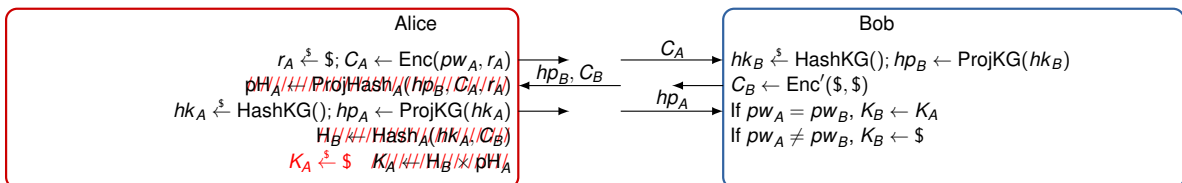
Send-queries to Bob: Non Oracle-Generated C_A



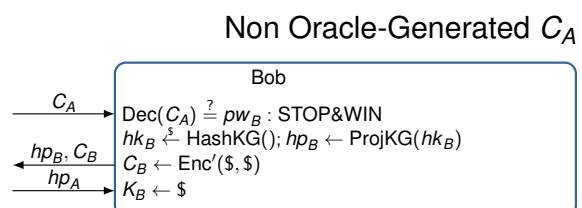
- The adversary must encrypt the correct password: password-guessing probability
- Smoothness
- IND-CPA

GL-PAKE: Security Proof

Send-queries to Alice: Oracle-Generated C_B

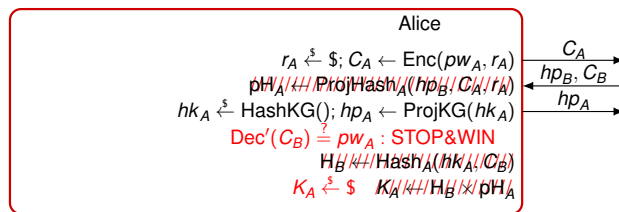


Smoothness



GL-PAKE: Security Proof

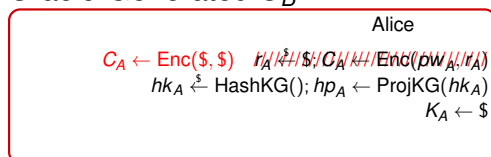
Send-queries to Alice: Non Oracle-Generated C_B



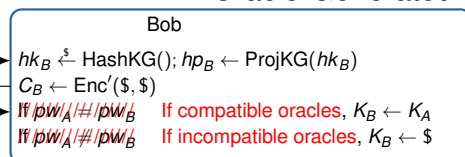
- The adversary must encrypt the correct password: password-guessing probability
- Smoothness

GL-PAKE: Security Proof

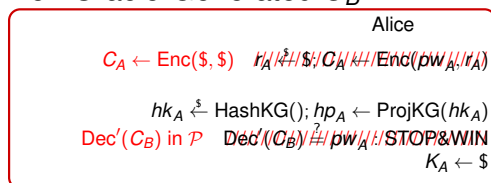
Oracle-Generated C_B



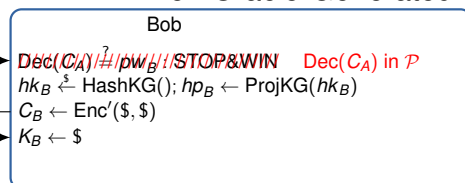
Oracle-Generated C_A



Non Oracle-Generated C_B



Non Oracle-Generated C_A



- 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 \mathcal{P} : the probability is q_S/N

Encryption schemes:

- (Enc, 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**
- (Enc', 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)

Labeled Cramer-Shoup Ciphertext Languages

Cramer-Shoup Encryption Scheme is an L-IND-CCA PKE:

$$C = (u_1 = g_1^r, u_2 = g_2^r, e = h^r m, v = (cd^t)^r) \text{ with } t = \mathcal{H}(\ell, u_1, u_2, e)$$

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^t)

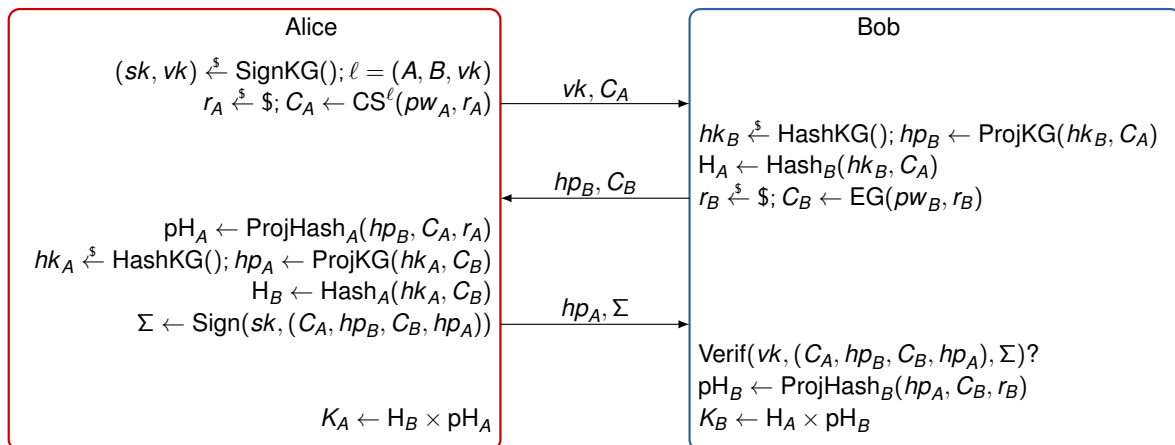
$$\begin{aligned} \text{HashKG}() : hk &= (\alpha, \beta, \gamma, \delta) \xleftarrow{\$} \mathbb{Z}_q^4 & \text{ProjKG}(hk, C) : hp &= g_1^\alpha g_2^\beta h^\gamma (cd^t)^\delta \\ \text{Hash}(hk, C) : H &= u_1^\alpha u_2^\beta (e/pw)^\gamma v^\delta & \text{ProjHash}(hp, C, r) : pH &= hp^r \end{aligned}$$

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

[Gennaro-Lindell – Eurocrypt '03]

GL-PAKE: Complete Protocol

[Gennaro-Lindell – Eurocrypt '03]



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

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: Explicit Proof

We denote $\mathcal{L}_A/\mathcal{L}_B$ the languages of the commitments C of pw_A/pw_B

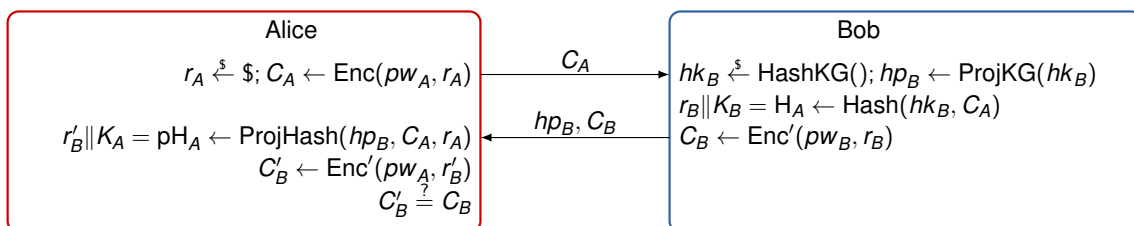
- Alice sends C_A , a commitment of pw_A with random coins r_A , to Bob
- Bob can ask to verify that $C_A \in \mathcal{L}_B$:
 - Bob sends hp_B to Alice, and computes $H_A \leftarrow \text{Hash}_B(hk_B, C_A)$
 - Alice can compute $pH_A \leftarrow \text{ProjHash}_A(hp, C_A, r_A)$
- Alice parses pH_A as $r'_B \| K_A$
- 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 $C'_B \stackrel{?}{=} C_B$

For a non-trivial language, the commitment C_A must be **perfectly binding**

To avoid false positive on $C'_B \stackrel{?}{=} C_B$, the commitment C_B must be **perfectly binding**
e.g., Public-Key Encryption Scheme

GK-PAKE

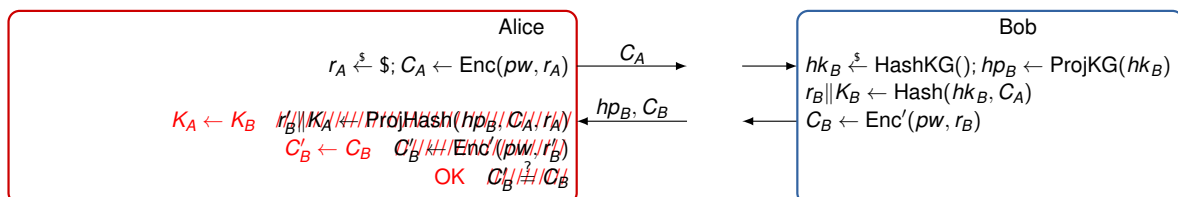
[Groce-Katz – CCS '10]



Which are the security properties of the encryption schemes ?

GK-PAKE: Security Proof

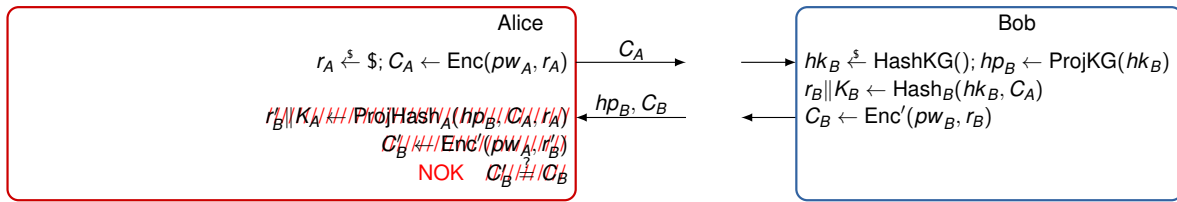
Send-query to Alice: Oracle-Generated C_B with $pw_B = pw_A = pw$



- C_B must be specific to this execution
- Oracle-Generated C_B must imply Oracle-Generated hp_B
- Correctness

GK-PAKE: Security Proof

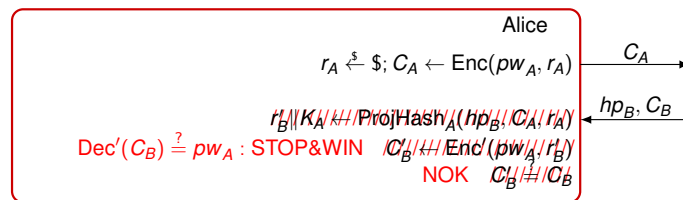
Send-query to Alice: Oracle-Generated C_B with $pw_B \neq pw_A$



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

GK-PAKE: Security Proof

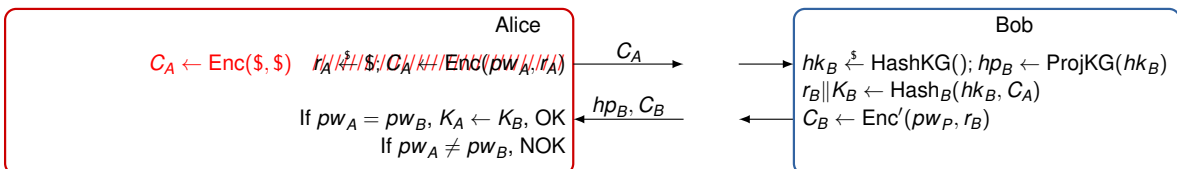
Send-query to Alice: Non Oracle-Generated C_B



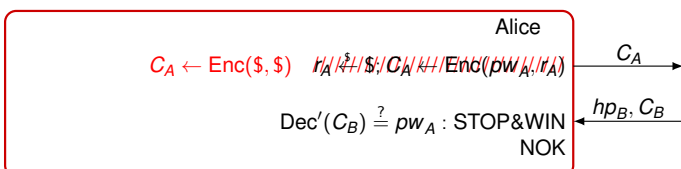
- C_B specific to this execution, and non-malleable
- The adversary must encrypt the correct password: password-guessing probability
- $\text{Dec}'(C_B) \neq pw_A \implies C'_B \neq C_B$

GK-PAKE: Security Proof

Send-query to Alice: Oracle-Generated C_B



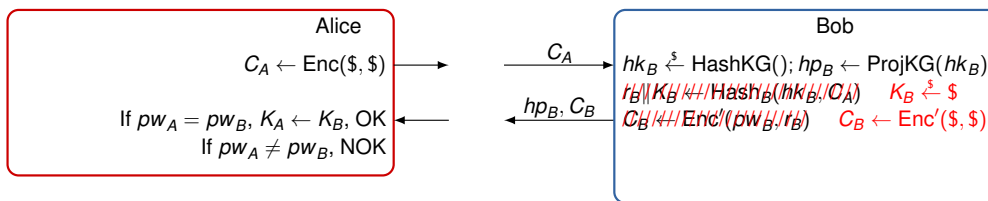
Send-query to Alice: Non Oracle-Generated C_B



IND-CPA

GK-PAKE: Security Proof

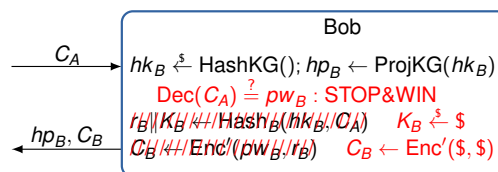
Send-query to Bob: Oracle-Generated C_A



- Correctness + IND-CCA

GK-PAKE: Security Proof

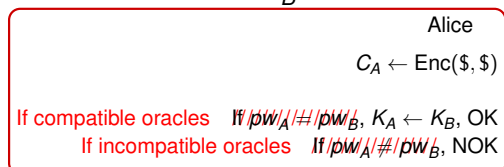
Send-query to Bob: Non Oracle-Generated C_A



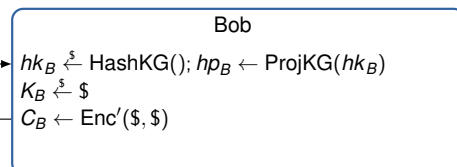
- The adversary must encrypt the correct password: password-guessing probability
- Smoothness + IND-CCA

GK-PAKE: Security Proof

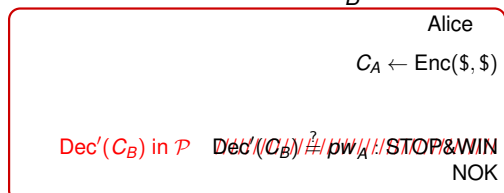
Oracle-Generated C_B



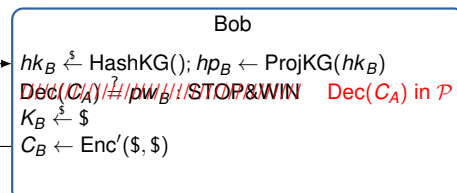
Oracle-Generated C_A



Non Oracle-Generated C_B



Non Oracle-Generated C_A



- No abort anymore: difference if the guesses are correct

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:

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

GK-PAKE

[Groce-Katz – CCS '10]

- Alice generates and sends $C_A = (u \leftarrow g^{r_A}, e \leftarrow h^{r_A} g^{pw_A}) \in \mathbb{G}^2$
- Bob
 - generates $hk_B = (\alpha, \beta)$ and $hp_B = g^\alpha h^\beta$
 - computes $r_B \| K_B = \text{KDF}(u^\alpha \cdot (e/g^{pw_B})^\beta)$
 - generates $C_B = (u_1 = g^{r_1}, u_2 = g^{r_2}, e = h^{r_B} g^{pw_B}, v = (cd^t)^{r_B})$,
with $t = \mathcal{H}(A, B, C_A, hp_B, u_1, u_2, e)$
 - sends $(hp_B, C_B) \in \mathbb{G}^5$
- Alice
 - computes $r'_B \| K_A = \text{KDF}(hp_B^{r'_A})$
 - generates $C'_B = (u'_1 = g^{r'_1}, u'_2 = g^{r'_2}, e = h^{r'_B} g^{pw_A}, v' = (cd^{t'})^{r'_B})$,
with $t' = \mathcal{H}(A, B, C_A, hp_B, u'_1, u'_2, e')$
 - 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

Better Efficiency

IND-PCA

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

- One either decrypts every C into pw and checks whether $pw \in \mathcal{P}$ or not
 \implies need of decryption oracle
- Or one checks for every $pw \in \mathcal{P}$ 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 χ

With hp possibly sent before χ : 1-round protocol with **KV-SPHF**

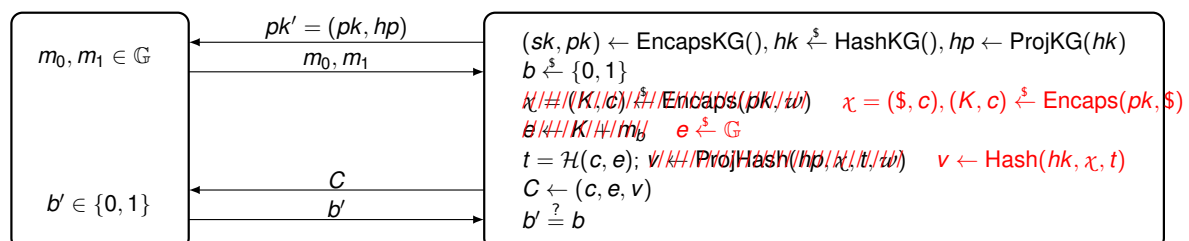
IND-PCA Encryption from KEM

$$\begin{aligned} \text{KeyGen}() : & (sk, pk) \xleftarrow{\$} \text{EncapsKG}() \\ & hk \xleftarrow{\$} \text{HashKG}(), hp \leftarrow \text{ProjKG}(hk) \\ & sk' = (sk, hk), pk' = (pk, hp) \end{aligned}$$

$$\begin{aligned} \text{Enc}(pk' = (pk, hp), m) : & (K, c) \xleftarrow{\$} \text{Encaps}(pk, r), e \leftarrow m + K \\ & \chi = (K, c), w = r \\ & t = \mathcal{H}(c, e), v = \text{ProjHash}(hp, \chi, t, w) \\ & C = (c, e, v) \end{aligned}$$

$$\begin{aligned} \text{Dec}(sk' = (sk, hk), C = (c, e, v)) : & K \leftarrow \text{Decaps}(sk, c), m \leftarrow e - K \\ & \chi = (K, c), t \leftarrow \mathcal{H}(c, e), v' \leftarrow \text{Hash}(hk, \chi, t) \\ & \text{If } (v' \neq v) \implies \text{Reject} \\ & \text{Else Return } m \end{aligned}$$

IND-PCA Security Proof



$$\text{PCA}(sk', m', C') : K \leftarrow \text{Decaps}(sk, c'), m' \leftarrow e' - K'$$

$$\begin{aligned} sk' = (sk, hk) \quad sk' = (hk) \quad \chi' \leftarrow (e' - m', c'), t' \leftarrow \mathcal{H}(c', e') \quad C' \neq C \implies (\chi', t') \neq (\chi, t) \\ C' = (c', e', v') \quad v'' \leftarrow \text{Hash}(hk, \chi', t') \\ v'' \neq v' \implies \text{Reject} \end{aligned}$$

Smoothness (soundness): $v'' = v' \implies e' - m'$ valid key $\implies m'$ valid plaintext
 + 2-Universal Smoothness (simulation-soundness)

Correctness + Indistinguishability/Hard Subset Membership $\implies \Pr[b' = b] = \frac{1}{2}$

$$\begin{aligned} \text{KeyGen}() : & \quad sk = (s, hk_1 = (x_1, x_2), hk_2 = (y_1, y_2)) \xleftarrow{\$} \mathbb{Z}_q^5 \\ & \quad pk = (g_2 \leftarrow g_1^s, hp_1 \leftarrow g_1^{x_1} g_2^{x_2}, hp_2 \leftarrow g_1^{y_1} g_2^{y_2}) \\ \text{Enc}(pk, m) : & \quad r \xleftarrow{\$} \mathbb{Z}_q; u = g_1^r, e \leftarrow g_2^r \times m \\ & \quad v = (hp_1 \times hp_2^t)^r, \text{ with } t \leftarrow \mathcal{H}(u, e) \\ \text{Dec}(sk, (u, e, v)) : & \quad m = e/u^s, v \stackrel{?}{=} u^{x_1+tx_2} \times (e/m)^{x_2+ty_2}, \text{ with } t \leftarrow \mathcal{H}(u, e) \end{aligned}$$

Theorem (IND-PCA Security)

This SCS encryption scheme is IND-PCA under the DDH assumption

GK-SPOKE (Simple Password-Only Key Exchange)

[Abdalla-Benhamouda-P. – PKC '15]

- Alice generates and sends $C_A = (u \leftarrow g^{r_A}, e \leftarrow h^{r_A} g^{pw_A}) \in \mathbb{G}^2$
- Bob
 - generates $hk_B = (\alpha, \beta)$ and $hp_B = g^\alpha h^\beta$
 - computes $r_B \| K_B = \text{KDF}(u^\alpha \cdot (e/g^{pw_B})^\beta)$
 - generates $C_B = (u' = g_1^{r'_B}, e' = g_2^{r'_B} g^{pw_B}, v = (cd^t)^{r'_B})$,
with $t = \mathcal{H}(A, B, C_A, hp_B, u, e)$
 - sends $(hp_B, C_B) \in \mathbb{G}^4$
- Alice
 - computes $r'_B \| K_A = \text{KDF}(hp_B^{r'_A})$
 - generates $C'_B = (u' = g_1^{r'_B}, e' = g_2^{r'_B} g^{pw_A}, v' = (cd^t)^{r'_B})$,
with $t' = \mathcal{H}(A, B, C_A, hp_B, u', e')$
 - 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

[Gennaro-Lindell – Eurocrypt '03]

Alice

- Alice sends C_A , a commitment of pw_A , to Bob
- Bob can ask to verify that $C_A \in \mathcal{L}_B$ (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 $C_B \in \mathcal{L}_A$ (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 $pH_B \leftarrow \text{ProjHash}_B(hp_A, C_B, w_B)$

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

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

KV-Smoothness

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

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

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

KV-SPHF for SCS Ciphertexts of m

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

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

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

KV-SPOKE (Simple Password-Only Key Exchange)

- Alice generates
 - $hk_A = (\alpha, \beta, \gamma, \delta)$ and $hp_A = (hp_1 \leftarrow g^\alpha h^\gamma c^\delta, hp_2 \leftarrow g^\beta d^\delta)$
 - $C_A = (u \leftarrow g^{r_A}, e \leftarrow h^{r_A} g^{pw_A}, v \leftarrow (cd^{t_A})^{r_A})$ 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
 - $t_B = \mathcal{H}(B, A, u', e', hp_B)$
 - $H_B = u'^{\alpha+t_B\beta} (e'/pw_A)^\gamma v'^\delta$
 - $pH_A = (hp'_1 hp'_2^{t_A})^{r_A}$
 - $K_A = pH_A \times H_B$

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

Outline

Introduction

1 Game-based Security

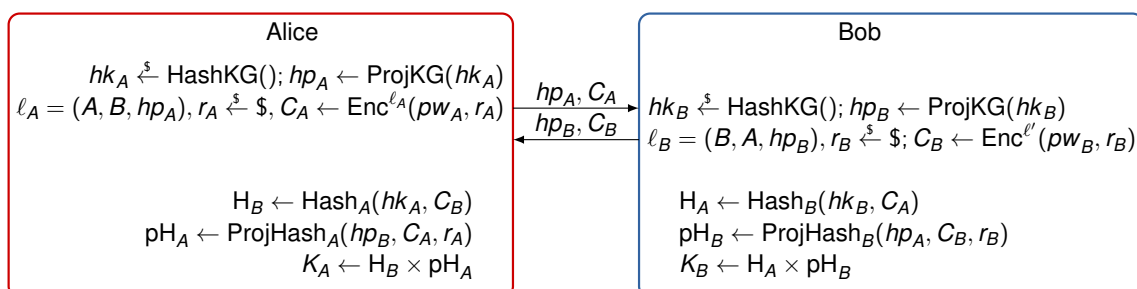
- Gennaro-Lindell PAKE
- Groce-Katz PAKE
- Improvements

2 Universal Composability

- UC-Secure PAKE: Static Corruptions
- UC-Secure PAKE: Adaptive Corruptions

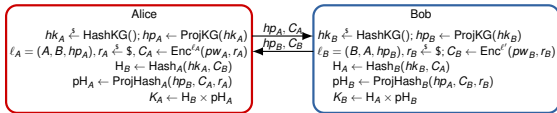
Conclusion

SPHF-Based PAKE: Protocol



UC-secure against static corruptions

SPHF-Based PAKE: Simulation



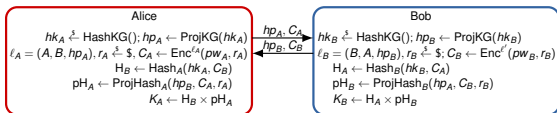
NewSession: for U with U'

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

Flow hp', C'

- Oracle-Generated from U' : $hk', hp' \leftarrow \text{ProjKG}(hk'), C' \leftarrow \text{Enc}^{\ell'}(pw', r')$
 - $H \leftarrow \text{Hash}(hk, C')$
 - ~~$pH \leftarrow \text{ProjHash}(hp', C, r)$~~ $pH \leftarrow \text{Hash}(hk', C)$
 - $K \leftarrow H \times pH$ Passwords known for corrupted players
- Non Oracle-Generated: $pw' \leftarrow \text{Dec}^{\ell'}(C')$ and $\text{TestPwd}(pw')$
 - ~~$H \leftarrow \text{Hash}(hk, C')$~~ $H \xleftarrow{\$} \$$ if incorrect guess
 - $pH \leftarrow \text{ProjHash}(hp', C, r)$
 - ~~$K \leftarrow H \times pH$~~ $K \xleftarrow{\$} \$$ if incorrect guess

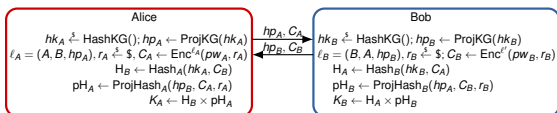
SPHF-Based PAKE: Simulation



Passwords known for corrupted players

- NewSession: for U with U'
 - $hk \xleftarrow{\$} \text{HashKG}(); hp \leftarrow \text{ProjKG}(hk)$
 - $\ell = (U, U', hp), r \xleftarrow{\$} \$, C \leftarrow \text{Enc}^\ell(pw, r)$ $C \leftarrow \text{Enc}^\ell(\$)$
- Flow hp', C'
 - Oracle-Generated from U' : $hk', hp' \leftarrow \text{ProjKG}(hk'), C' \leftarrow \text{Enc}^{\ell'}(pw', r')$
 - ~~$H \leftarrow \text{Hash}(hk, C')$~~ $H \leftarrow \$$
 - ~~$pH \leftarrow \text{ProjHash}(hp', C, r)$~~ $pH \leftarrow H'$, if $pw' = pw$; $pH \xleftarrow{\$} \$$, if $pw' \neq pw$
 - ~~$K \leftarrow H \times pH$~~ K Same $\$$ as compatible partner, independent $\$$ if incompatible partner
 - Non Oracle-Generated: $pw' \leftarrow \text{Dec}^{\ell'}(C')$ and $\text{TestPwd}(pw')$
 - $K \xleftarrow{\$} \$$ if incorrect guess / possibly chosen by the adversary otherwise

SPHF-Based PAKE: Simulation



- NewSession: for U with U'
 - $hk \xleftarrow{\$} \text{HashKG}(); hp \leftarrow \text{ProjKG}(hk)$
 - $\ell = (U, U', hp), C \leftarrow \text{Enc}^\ell(\$)$
- Flow hp', C'
 - Oracle-Generated from U' : $hk', hp' \leftarrow \text{ProjKG}(hk'), C' \leftarrow \text{Enc}^{\ell'}(pw', r')$
 - Same $\$$ as compatible partner, independent $\$$ if incompatible partner NewKey
 - Non Oracle-Generated: $pw' \leftarrow \text{Dec}^{\ell'}(C')$ and $\text{TestPwd}(pw')$
 - ~~$K \leftarrow H \times pH$~~ $K \xleftarrow{\$} \$$ if incorrect guess // possibly chosen by the adversary otherwise NewKey

Introduction

1 Game-based Security

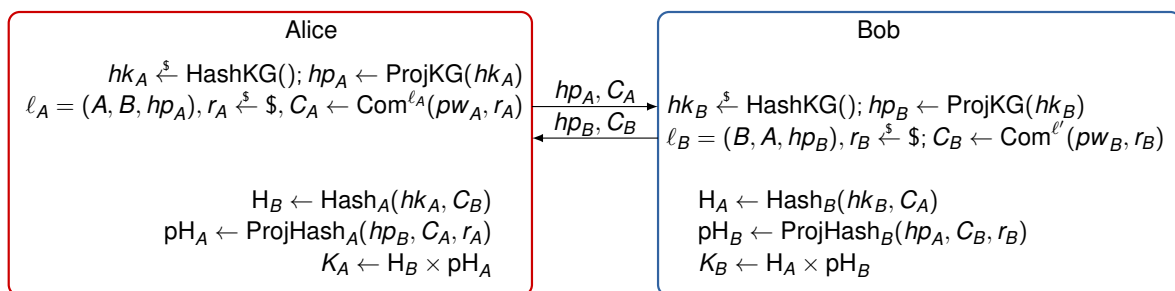
- Gennaro-Lindell PAKE
- Groce-Katz PAKE
- Improvements

2 Universal Composability

- UC-Secure PAKE: Static Corruptions
- UC-Secure PAKE: Adaptive Corruptions

Conclusion

SPHF-Based PAKE: Protocol



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

⇒ no security against **adaptive corruptions**

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

⇒ security against **adaptive corruptions**

SPHF-Friendly Commitments

- Based on Cramer-Shoup (extractability) and Pedersen (equivocability)

Inspired from the Canetti-Fischlin commitment

[Canetti-Fischlin – Crypto '01]

⇒ Commitment size linear in mk^2

[Abdalla-Chevalier-P. – Crypto '09]

- Improvement with Haralambiev (equivocability)

[Haralambiev – PhD Thesis '11]

⇒ Commitment size linear in mk

[Abdalla-Benhamouda-Blazy-Chevalier-P. – Asiacrypt '13]

- SPHF-Friendly variant of FLM commitment

[Fischlin-Libert-Manulis – Asiacrypt '11]

⇒ Commitment size linear in k

[Blazy-Chevalier – Asiacrypt '16]

m = length of the password k = security parameter

■ 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**
the **GL methodology** widely used for PAKE

[Katz-Ostrovsky-Yung – Crypto '01]

[Gennaro-Lindell – Eurocrypt '03]

- BPR-secure protocols
- **UC-secure** protocols for static corruptions [Canetti-Halevi-Katz-Lindell-MacKenzie – Eurocrypt '05]
- 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