

Intercambio de claves de Diffie–Hellman

Clase 14

<https://apuntes.indcpa.com>

Versión 0.0.1, junio 2024

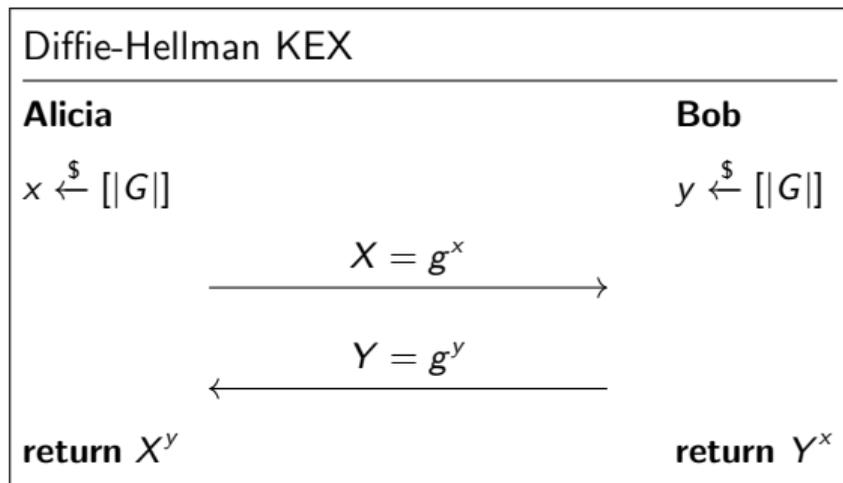
- Hasta 1970, toda la criptografía utilizada era de clave simétrica.
- Esto no era una gran limitación dado que el uso principal era militar, donde “canales seguros” de intercambio de clave físicos existen.
- Esta es una gran limitación para uso civil.
- Vamos ahora a introducir brevemente el intercambio de claves (KEX, “key exchange”) de Diffie y Hellman.
- Lo hacemos por motivos informacionales e históricos, al ser el primer protocolo de PKC considerado seguro.
- No vamos a intentar demostrar su seguridad en detalle, porque los modelos de seguridad de KEX pueden ser muy complicados.

<pre> Exp_{PAKE,u,t,A}: pp_{AKE} ← AKE.Setup For i ∈ [μ]: (pk_i, sk_i) ← AKE.Gen(pp_{AKE}, P_i); crp_i := false //Corruption variable PKList := {pk_i}_{i∈[μ]}; b ← {0, 1} For (i, s) ∈ [μ] × [t]: var_i := (st_i[*], Pid_i[*], k_i[*], Φ_i[*]) := (0, 0, 0, 0); Aflag_i[*] := false //Whether Pid_i[*] is corrupted when π_i[*] accepts T_i[*] := false; kReve_i[*] := false // Test, Key Reveal variables stReve_i[*] := false, FirstAcc_i[*] := 0 // State Reveal & First Acceptance variables b* ← A^{PAKE(1)}(pp_{AKE}, PKList) Wins_{adv} := false Wins_{adv} := true, If ∃(i, s) ∈ [μ] × [t] s.t. (1) Φ_i[*] = accept //π_i[*] is τ-accepted (2) Aflag_i[*] = false //P_i is τ-corrupted with j := Pid_i[*] and τ > τ (3) (3.1) ∨ (3.2) ∨ (3.3). Let j := Pid_i[*] (3.1) ∃ t ∈ [t] s.t. Partner(π_i[*] ← π_j[*]) (3.2) ∃ t ∈ [t], (j', t') ∈ [μ] × [t] with (j, t) ≠ (j', t') s.t. Partner(π_i[*] ← π_j[*]) ∧ Partner(π_i[*] ← π_{j'}[*]) (3.3) ∃ t ∈ [t], (i', s') ∈ [μ] × [t] with (i, s) ≠ (i', s') s.t. Partner(π_i[*] ← π_j[*]) ∧ Partner(π_i[*] ← π_{j'}[*]) //Replay attacks Wins_{adv} := false If b* = b: Wins_{adv} := true; Return 1 Else: Return 0 Partner(π_i[*] ← π_j[*]): //Checking whether Partner(π_i[*] ← π_j[*]) If π_i[*] sent the first message and k_i[*] = K(π_i[*], π_j[*]) ≠ ∅: Return 1 If π_j[*] received the first message and k_j[*] = K(π_j[*], π_i[*]) ≠ ∅: Return 1 Return 0 π_i[*](msg, j): //π_i[*] executes AKE according to the protocol specification If Pid_i[*] = ∅: Pid_i[*] := j If Pid_i[*] = j: π_i[*] receives msg and uses res, var_i[*] to generate the next message msg_i[*] of AKE, and updates (st_i[*], Pid_i[*], k_i[*], Φ_i[*]): If msg = ⊥: π_i[*] generates the first message msg_i[*] as initiator; If msg is the last message of AKE: msg_i[*] := ∅; Return msg_i[*] If Pid_i[*] ≠ j: Return ⊥ C_{AKE}(query): If query=RegisterCorrupt(u, pk_u): If u ∈ [μ]: Return ⊥ PKList := PKList ∪ {pk_u} crp_u := true Return PKList </pre>	<pre> C_{AKE}(query): If query=Send(i, s, j, msg): If Φ_i[*] = accept: Return ⊥ msg_i[*] ← π_i[*](msg, j) If Φ_i[*] = accept: If crp_i = true: Aflag_i[*] := true; // Determine whether π_i[*] accepts before its partner If crp_i = false ∧ ∃ t ∈ [t] s.t. Partner(π_i[*] ← π_t[*]): If Φ_t[*] ≠ accept: FirstAcc_t[*] := true; FirstAcc_i[*] := false If Φ_t[*] = accept: FirstAcc_t[*] := false; FirstAcc_i[*] := true Return msg_i[*] If query=Corrupt(i): If i ∉ [μ]: Return ⊥ For s ∈ [t]: If FirstAcc_i[*] = false ∧ stReve_i[*] = true: If T_i[*] = true: Return ⊥; //avoid TA6 If ∃ t ∈ [t] s.t. Partner(π_i[*] ← π_t[*]): If T_t[*] = true: Return ⊥; //avoid TA7 crp_i := true Return sk_i If query=SessionKeyReveal(i, s): If Φ_i[*] ≠ accept: Return ⊥ If T_i[*] = true: Return ⊥ //avoid TA2 Let j := Pid_i[*] If ∃ t ∈ [t] s.t. Partner(π_i[*] ↔ π_t[*]): If T_t[*] = true: Return ⊥; //avoid TA4 kReve_i[*] := true; Return k_i[*] If query=StateReveal(i, s): If FirstAcc_i[*] = false ∧ crp_i = true: If T_i[*] = true: Return ⊥; //avoid TA6 Let j := Pid_i[*] If ∃ t ∈ [t] s.t. Partner(π_i[*] ← π_t[*]): If T_t[*] = true: Return ⊥; //avoid TA7 stReve_i[*] := true; Return st_i[*] If query=Test(i, s): If Φ_i[*] ≠ accept ∨ Aflag_i[*] = true ∨ kReve_i[*] = true ∨ T_i[*] = true: Return ⊥ //avoid TA1, TA2, TA3 If FirstAcc_i[*] = false: If crp_i = true ∧ stReve_i[*] = true: Return ⊥ //avoid TA6 Let j := Pid_i[*] If ∃ t ∈ [t] s.t. Partner(π_i[*] ↔ π_t[*]): If kReve_t[*] = true ∨ T_t[*] = true: Return ⊥ //avoid TA4, TA5 If ∃ t ∈ [t] s.t. Partner(π_i[*] ↔ π_t[*]): If FirstAcc_t[*] = false ∧ crp_t = true ∧ stReve_t[*] = true: Return ⊥ //avoid TA7 T_i[*] := true; k₀ := k_i; k₁ ← K; Return k₀ </pre>
--	---

Fig. 5. The security experiments $\text{Exp}_{\text{PAKE},u,t,A}$ (both without red text) and $\text{Exp}_{\text{PAKE},u,t,A}^{\text{reply}}$ (with red text). The list of trivial attacks is given in Table 2.

Modelo de seguridad de intercambio de clave autenticado usado en Han *et al.*, CRYPTO 2021.

Supongamos que dos partes Alicia y Bob se ponen de acuerdo públicamente sobre un grupo cíclico finito \mathbb{G} y sobre un generador g , tal que $\mathbb{G} = \langle g \rangle$.

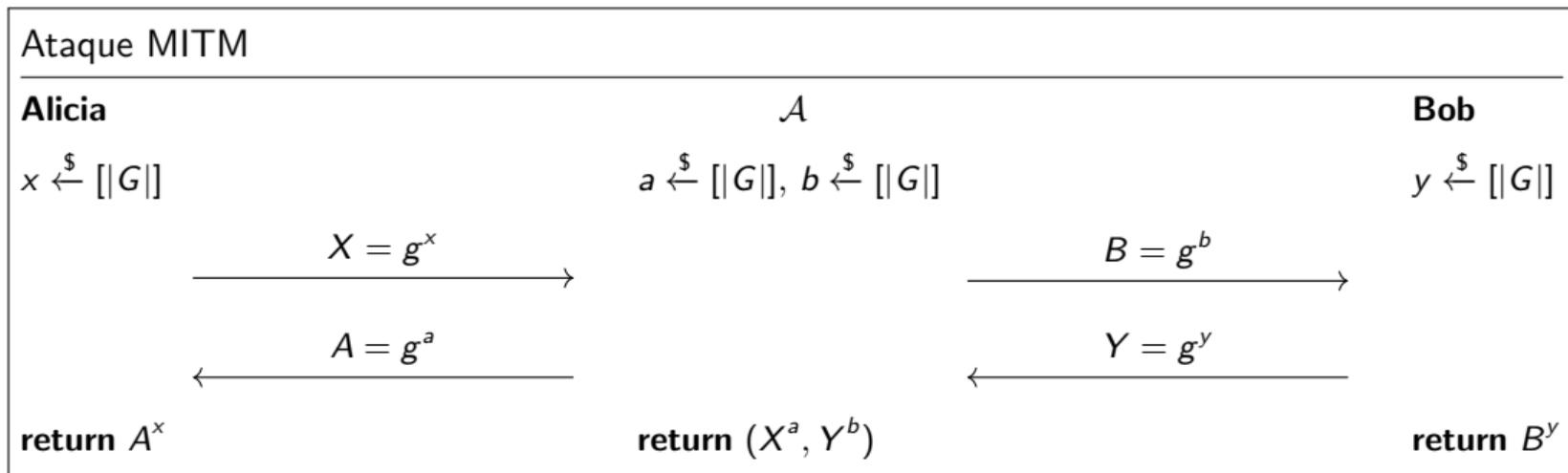


- Notamos que $X^y = (g^x)^y = g^{xy} = g^{yx} = (g^y)^x = Y^x$.
 - ▶ Alicia y Bob retornan el mismo elemento $g^{xy} \in \mathbb{G}$.
- Podemos definir una función de hash $H: \mathbb{G} \rightarrow \{0, 1\}^\ell$, y utilizar $H(g^{xy})$ como clave para cifrado (simétrico) autenticado.
 - ▶ Formalmente, se precisa que H sea una función de derivación de claves (KDF, key-derivation function).

Definir la seguridad del KEX puede ser complicado. Supongamos un adversario que se limita a observar pasivamente la comunicacion (el “transcript”) (g^x, g^y) .

- Si CDH es difícil en \mathbb{G} , sabemos que calcular g^{xy} es difícil.
- Es esto suficiente a garantizar que DH es seguro frente a un adversario “pasivo” (que solo observa trafico de red)? CDH no excluye que “mitad de los bits” de g^{xy} sean aprendibles. Esto podria teoricamente resultar en vulnerar la siguiente fase de comunicacion.
 - ▶ Junto a un hash, esto es suficiente en el Random Oracle Model, *c.f.* § 11.5.1 **BS**.
- De aqui la idea de usar DDH: si (g^x, g^y, g^{xy}) es indistinguible de (g^x, g^y, g^z) con z aleatorio, \mathcal{A} aprende “nada” sobre xy observando (g^x, g^y) .

- Notamos también que \mathcal{A} podría no limitarse simplemente a observar tráfico, y también interceptarlo si tiene control sobre la red.
- Este tipo de ataque se llama machine-in-the-middle (MITM).



- En este caso \mathcal{A} podría fingir de ser Bob a Alice, y Alicia a Bob.
- Por ejemplo, podría interceptar $\text{Enc}(H(g^{ax}), m)$ de Alicia a Bob, decifrarlo, volverlo a cifrar como $\text{Enc}(H(g^{by}), m)$, y finalmente mandarlo a Bob. Alicia y Bob no podrían detectar el ataque.

- Este tipo de tactica no funciona si Alice y Bob intercambian claves secretas de persona, y utilizan exclusivamente criptografia simetrica.
- En el setting de criptografia de clave publica, defendernos de este tipo de tactica requiere nuevos instrumentos, para construir una “infraestructura de claves publicas” (PKI, public-key infrastructure).
 - ▶ En lugar de iniciar intercambio de claves en el momento, usaremos esquemas de cifrado de clave publica (PKE, “public-key encryption”), donde Bob primero publica una clave pk, y Alicia cifra usando pk.
 - ▶ La clave de Bob va a ser “certificada” como autentica usando un sistema de firma digital.
 - ▶ Con estas tecnicas, podremos verificar la identidad de un poseedor de clave publica, evitando que A nos engañe en cifrar hacia una clave que no le pertenece a Bob.

Empezamos entonces definiendo y construyendo PKE.