

Password Authenticated Key Exchange

Martin Stanek

Department of Computer Science
Comenius University
`stanek@dcs.fmph.uniba.sk`

Cryptology 1 (2020/21)

Motivation

- ▶ authenticate user/client using a password
- ▶ common scenario for authentication in web application:
 - ▶ TLS, server authentication, secure channel
 - ▶ username/password login form, server verifies submitted password
- ▶ some problems with this approach ...
- ▶ phishing attacks – login to fake web site
 - ▶ attacker gets all authentication data (username, password)
 - ▶ multi-factor authentication can mitigate the risk
- ▶ TLS might not be available
- ▶ PAKE – Password Authenticated Key Exchange (agreement)

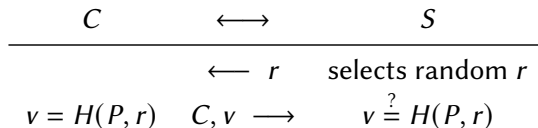
Goal: (mutual) authentication of two or more parties and establishing keys for subsequent communication

Passwords

- ▶ special type of shared secret
- ▶ easy to use
- ▶ potential problems: guessing (low entropy), brute-force attack
 - ▶ limited length (“small” set of possible passwords)
 - ▶ passwords from various dictionaries
 - ▶ patterns/non-uniform selection of passwords

Simple authentication protocol

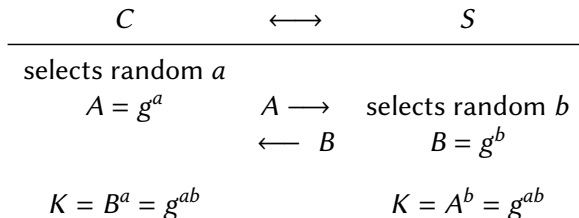
- ▶ challenge/response protocol
- ▶ (+) password not transmitted in plaintext
- ▶ notation: password P , hash function H



- ▶ drawbacks:
 - ▶ one way authentication (only C is authenticated)
 - ▶ attacker can accept any v and continue the session with C
 - ▶ MITM attack: attacker relays communication between C and S
 - ▶ no session key agreed in the protocol

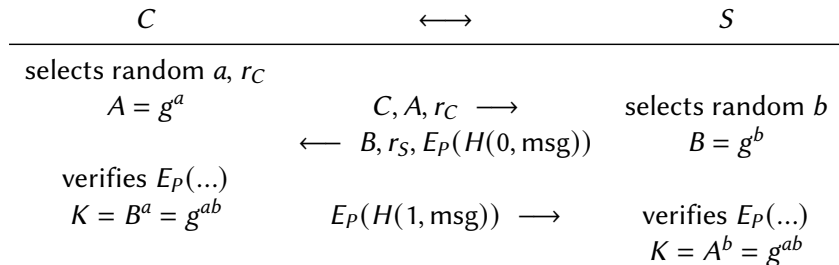
Simple key-agreement protocol

- ▶ Diffie-Hellman protocol (using a group where CDH is hard)
- ▶ MITM attack (cause: unauthenticated exchange of parameters)
- ▶ notation: generator g



Simple AKE protocol

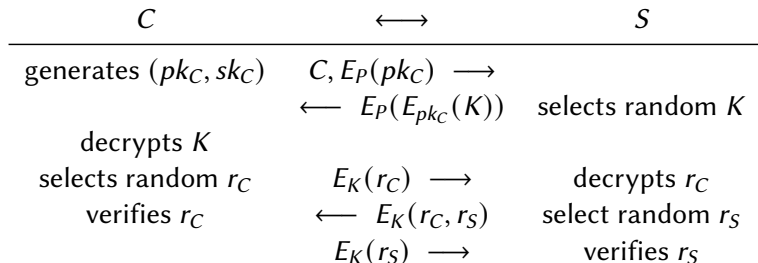
Goals: password never sent as a plaintext, authenticate both parties, agree on a session key, prevent MITM attack



- ▶ notation: $\text{msg} = C || A || B || r_C || r_S$; H is a hash function
- ▶ E_P – e.g. symmetric cipher or MAC_P , key is derived from P
- ▶ problem: offline dictionary attack – testing passwords offline using eavesdropped communication

EKE (Encrypted Key Exchange) – general description

- ▶ Bellare, Merritt (1992)
- ▶ first PAKE protocol
- ▶ prevents offline dictionary attack (and achieves previous goals as well)



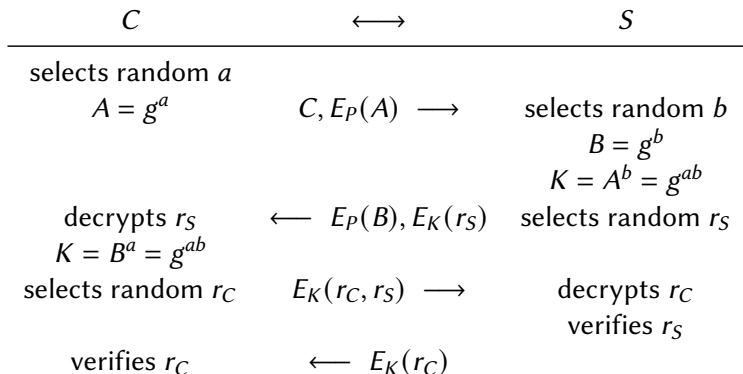
- ▶ notation: (pk_C, sk_C) pair of keys for asymmetric encryption; E_{pk_C} public-key encryption, E_P symmetric encryption using a key derived from P ; K session key

EKE remarks

- ▶ EKE is secure against offline dictionary attack, if all (or almost all) decryptions for distinct passwords yield
 - ▶ valid public keys for message in the first step
 - ▶ valid ciphertexts for message in the second step
- ▶ implementation problem – choosing suitable encryption schemes (symmetric and public-key)
- ▶ *partition attack*
 - ▶ offline attack
 - ▶ if decryption with P' yield an incorrect/impossible public key, then $P \neq P'$
 - ▶ example: RSA ... n with small factors, even e
 - ▶ multiple runs of the protocol \Rightarrow password is uniquely determined
 - ▶ E_P should not leak information about P

DH-EKE

- ▶ variant of EKE with DH protocol for key agreement
- ▶ only modular groups (!)
- ▶ this variant follows the original proposal (Bellare, Merritt, 1992):



DH-EKE remarks

- ▶ more refined version of the protocol is EAP-EKE (RFC 6124), e.g.
 - ▶ separate keys are derived for the protocol itself and for session
 - ▶ encryption with MAC used for messages containing nonces (here: r_C, r_S)
 - ▶ additional data are computed, using a key derived from the shared key and all messages up to given point – protects integrity of the negotiated parameters
 - ▶ explicit requirements for groups, e.g. g is a primitive element (generator) of the group, p is a “safe” prime
 - ▶ explicit list of suitable groups and their generators
- ▶ what if g is not a generator:
 - ▶ decrypt $E_{P'}(A)$ and $E_{P'}(B)$ using password P'
 - ▶ if a generator is obtained, P' is incorrect
- ▶ there is $\sim 50\%$ generators in groups with safe prime modulus, i.e. $q = 2q' + 1$ (where q' is a prime)

Problems with EKE (DH-EKE, EAP-EKE)

- ▶ server knows the password
- ▶ successful attack on server results in compromised passwords
- ▶ passwords are usually stored “salted”
 - ▶ after a breach the offline dictionary attack is always possible
 - ▶ we don't want to make it easier
- ▶ DH constructions are hard to translate to elliptic curves
 - ▶ How to ensure that decryption with wrong password yields a point on elliptic curve?

Secure Remote Password protocol (SRP)

- ▶ PAKE protocol, server does not store password in plaintext
 - ▶ other properties are preserved (prevention of offline dictionary attack etc.)
- ▶ original proposal: Thomas Wu (1998)
- ▶ RFC 2945 (2000) version SRP-3
- ▶ using SRP-6 (2002) together with TLS: RFC 5054 (2007)
- ▶ other standardization: IEEE P1363.2, ISO IEC 11770-4
- ▶ Apple uses SRP in iCloud, according *iOS Security* (2018):

The HSM cluster verifies that a user knows their iCloud Security Code using Secure Remote Password protocol (SRP); the code itself isn't sent to Apple.

Evolution of SRP: SRP-3

- ▶ T. Wu, *The Secure Remote Password Protocol*, 1998
- ▶ RFC 2945, *The SRP Authentication and Key Exchange System*
- ▶ protocol slightly differs in these documents (we will follow the first one)
 - ▶ explicit choice of random u vs. derivation of u from B
 - ▶ construction of the first verification message M_1
- ▶ calculation in $\text{GF}(n)$, where n is a large prime
 - ▶ both operations are used (“+” and “.”)
- ▶ notation:
 - ▶ g – generator of (\mathbb{Z}_n^*, \cdot)
 - ▶ password P
 - ▶ random salt s
 - ▶ hash function H
- ▶ P is stored on server as a verifier $v = g^x$, where $x = H(s, P)$

SRP-3 – protocol

C	\longleftrightarrow	S
<hr/>		
selects random a		
$A = g^a$	$C, A \longrightarrow$	selects random b, u
	$\longleftarrow s, B, u$	$B = v + g^b$
computes:		computes:
$x = H(s, P)$		$S = (Av^u)^b$
$S = (B - g^x)^{a+ux}$		$K = H(S)$
$K = H(S)$		
$M_1 = H(A, B, K)$	$M_1 \longrightarrow$	verifies M_1
verifies M_2	$\longleftarrow M_2$	verifies $M_2 = H(A, M_1, K)$

► computation of shared secret S :

- client: $(B - g^x)^{a+ux} = (g^x + g^b - g^x)^{a+ux} = g^{ab+ubx}$
- server: $(Av^u)^b = (g^a \cdot g^{xu})^b = g^{ab+ubx}$

SRP-3 – protocol

C	\longleftrightarrow	S
selects random a		
$A = g^a$	$C, A \longrightarrow$	selects random b, u
	$\longleftarrow s, B, u$	$B = v + g^b$
computes:		computes:
$x = H(s, P)$		$S = (Av^u)^b$
$S = (B - g^x)^{a+ux}$		$K = H(S)$
$K = H(S)$		
$M_1 = H(A, B, K)$	$M_1 \longrightarrow$	verifies M_1
verifies M_2	$\longleftarrow M_2$	verifies $M_2 = H(A, M_1, K)$

► computation of shared secret S :

- client: $(B - g^x)^{a+ux} = (g^x + g^b - g^x)^{a+ux} = g^{ab+ubx}$
- server: $(Av^u)^b = (g^a \cdot g^{xu})^b = g^{ab+ubx}$

SRP-3 – security goals

- ▶ assumption: active attacker with ability to eavesdrop and manipulate transmitted data
- ▶ What security goals does SRP have?
 - ▶ confidentiality of P and x
 - ▶ confidentiality of K
 - ▶ security against offline dictionary attack

SRP-3 – remarks (1)

- ▶ Why B depends on v ?
 - ▶ simpler alternative: $B = g^b$, C does not need to compute g^x , rest of the protocol intact
 - ▶ attacker E asks the server for s and then impersonates the server
 1. $C \rightarrow E(S)$: $C, A = g^a$
 2. $E(S) \rightarrow C$: $s, B = g^b, u$, for randomly selected b, u
 3. $C \rightarrow E(S)$: $M_1 = H(A, B, K)$, where $S = B^{a+ux}$ and $K = H(S)$
- ▶ now E can perform this offline dictionary attack:
 - ▶ E computes x', v' for a password P' and then computes $S' = (Av'^u)^b$ and $K' = H(S')$
 - ▶ if $P = P'$ then those values are equal to values computed by C
 - ▶ E verifies this with check $H(A, B, K') = M_1$
- ▶ “+ v ” prevents attack – the attacker can't use a single instance to test unlimited number of passwords (he must choose v' that C subtracts)
- ▶ Exercise: What is wrong with this modification?
 - ▶ use $B = v \cdot g^b$ and C computes $S = (B/g^x)^{a+ux}$
 - ▶ advantage: we work only in the group (\mathbb{Z}_n^*, \cdot)

SRP-3 – remarks (2)

- ▶ Why is u random, instead of some constant?
 - ▶ attacker E can impersonate C
 - ▶ assumptions: E obtains v and s (knowing v requires access to server's data)
 1. $E(C) \rightarrow S: C, A = g^a \cdot v^{-u}$
 2. $S \rightarrow E(C): s, B$, where $B = v + g^b$
 3. E computes: $S = (B - v)^a = g^{ab}$
 S computes: $S = (A \cdot v^u)^b = (g^a \cdot v^{-u} \cdot v^u)^b = g^{ab}$
 - ▶ therefore u must be unpredictable (known before C sends A)
- ▶ no proofs of security claims

SRP-3 – *two-for-one* password guessing attack

- ▶ neither x nor v are known to attacker
- ▶ online password guessing using interaction with C :
 - ▶ attacker E (knows s) guesses P' and computes $x' = H(s, P')$, $v' = g^{x'}$
 - ▶ E impersonates the server using these values x' , v'
 - ▶ if the protocol finishes successfully (M_1 is correct), then P' is correct
- ▶ guessing two passwords simultaneously:
 1. E make a guess P_1, P_2 and computes corresponding x_1, x_2 and v_1, v_2
 2. $C \rightarrow E(S)$: C, A
 3. $E(S) \rightarrow C$: $s, B = g^{x_1} + g^{x_2}, u$
 4. $C \rightarrow E(S)$: $M_1 = H(A, B, K)$, where $K = H(S) = H((B - g^x)^{a+ux})$
- ▶ value $S = (B - g^x)^{a+ux} = (g^{x_1} + g^{x_2} - g^x)^{a+ux}$
 - ▶ if $P = P_1$ (or $P = P_2$), then C computes $S_1 = g^{x_2(a+ux_1)}$ (or $S_2 = g^{x_1(a+ux_2)}$)
 - ▶ E can compute $S'_1 = (A \cdot v_1^u)^{x_2}$ a $S'_2 = (A \cdot v_2^u)^{x_1}$
 - ▶ if $P = P_1$: $S'_1 = (g^a \cdot g^{x_1 u})^{x_2} = g^{x_2(a+ux_1)} = S_1$
 - ▶ if $P = P_2$: $S'_2 = (g^a \cdot g^{x_2 u})^{x_1} = g^{x_1(a+ux_2)} = S_2$
 - ▶ E can decide if any of those cases happened using M_1
- ▶ E does not have to chose u in a special way, the attack works even if u is computed as a truncated $H(B)$ (RFC 2945)

SRP-3 – *two-for-one* password guessing attack

- ▶ neither x nor v are known to attacker
- ▶ online password guessing using interaction with C :
 - ▶ attacker E (knows s) guesses P' and computes $x' = H(s, P')$, $v' = g^{x'}$
 - ▶ E impersonates the server using these values x' , v'
 - ▶ if the protocol finishes successfully (M_1 is correct), then P' is correct
- ▶ guessing two passwords simultaneously:
 1. E make a guess P_1, P_2 and computes corresponding x_1, x_2 and v_1, v_2
 2. $C \rightarrow E(S)$: C, A
 3. $E(S) \rightarrow C$: $s, B = g^{x_1} + g^{x_2}, u$
 4. $C \rightarrow E(S)$: $M_1 = H(A, B, K)$, where $K = H(S) = H((B - g^x)^{a+ux})$
- ▶ value $S = (B - g^x)^{a+ux} = (g^{x_1} + g^{x_2} - g^x)^{a+ux}$
 - ▶ if $P = P_1$ (or $P = P_2$), then C computes $S_1 = g^{x_2(a+ux_1)}$ (or $S_2 = g^{x_1(a+ux_2)}$)
 - ▶ E can compute $S'_1 = (A \cdot v_1^u)^{x_2}$ a $S'_2 = (A \cdot v_2^u)^{x_1}$
 - ▶ if $P = P_1$: $S'_1 = (g^a \cdot g^{x_1 u})^{x_2} = g^{x_2(a+ux_1)} = S_1$
 - ▶ if $P = P_2$: $S'_2 = (g^a \cdot g^{x_2 u})^{x_1} = g^{x_1(a+ux_2)} = S_2$
 - ▶ E can decide if any of those cases happened using M_1
- ▶ E does not have to chose u in a special way, the attack works even if u is computed as a truncated $H(B)$ (RFC 2945)

SRP-6

- ▶ T. Wu, *SRP-6: Improvements and Refinements to the Secure Remote Password Protocol*, 2002
- ▶ motivation for new version:
 1. two-for-one attack (parameter k used as a multiplication factor for v)
 2. implementation problem with message order (when group parameters must be sent)
 - ▶ 1 additional round required
 - ▶ solution: parameters/group ID and B sent before A
 - ▶ A sent together with M_1
- ▶ parameter k
 - ▶ SRP-6: $k = 3$; SRP-6a: $k = H(n, g)$
 - ▶ without knowledge of $\text{dlog}_g k$ the two-for-one attack does not work
 - ▶ computation $k = H(n, g)$ makes harder malicious choice n, g , where the attacker knows $\text{dlog}_g k$

SRP-6 protocol (original message order)

C	\longleftrightarrow	S
<hr/>		
selects random a		
$A = g^a$	$C, A \longrightarrow$	selects random b
	$\longleftarrow s, B$	$B = kv + g^b$
computes:		computes:
$u = H(A, B)$		$u = H(A, B)$
$x = H(s, P)$		$S = (Av^u)^b$
$S = (B - kg^x)^{a+ux}$		$K = H(S)$
$K = H(S)$		

► computation of shared secret S :

- client: $(B - kg^x)^{a+ux} = (kg^x + g^b - kg^x)^{a+ux} = g^{ab+ubx}$
- server: $(Av^u)^b = (g^a \cdot g^{xu})^b = g^{ab+ubx}$

SRP-6 protocol (cont.)

- ▶ additional messages for verifying K (equality on both ends):

C	\longleftrightarrow	S
$M_1 = H(H(n) \oplus H(g), H(C), s, A, B, K)$	$M_1 \longrightarrow$	verifies M_1
verifies M_2	$\longleftarrow M_2$	$M_2 = H(A, M_1, K)$

SRP remarks (1)

- ▶ S send s to anyone
 - ▶ salt is not secret, however ...
 - ▶ knowing s allows a pre-computation (before obtaining v), e.g. constructing TMTD tables \Rightarrow pre-computation attack
- ▶ protocol uses multiplication and addition
 - ▶ group operation is not enough
 - ▶ can't be translated to elliptic curves (less efficient)
- ▶ specific requirements for n and g ("safe prime" and generator)
 - ▶ direct use of some standardized parameters if not possible
 - ▶ RFC 5054 defines specific 1024, 1536 a 2048-bit primes and generators
 - ▶ larger primes are adopted from RFC 3526 (More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)), but with different g (generator)

SRP remarks (2)

- ▶ What if g is not a generator?
 - ▶ g generates a proper subgroup $[g]$ of (\mathbb{Z}_n^*, \cdot)
 - ▶ if for some P' the value $B - v' = B - g^{H(s, P')} \notin [g]$, then P' is not correct password \Rightarrow partition attack

Conclusion

- ▶ many PAKE protocols exist
- ▶ balanced PAKE protocols (both parties know the password):
 - ▶ EKE, DH-EKE, Dragonfly (SAE), SPEKE, J-PAKE, ...
- ▶ augmented, or asymmetric PAKE protocols (client/server)
 - ▶ server does not store password-equivalent data (i.e. data that allow successful authentication as a client)
 - ▶ SRP, Augmented-EKE, B-SPEKE, OPAQUE, ...
- ▶ first protocol resistant to pre-computation attack: OPAQUE (2018)

OPAQUE

- ▶ PAKE secure against pre-computation attack
- ▶ main idea:
 - ▶ combination of OPRF and AKE protocol, or
 - ▶ combination of OPRF and PAKE protocol
 - ▶ AKE and PAKE must have suitable properties (they can't be arbitrary)
- ▶ OPRF (Oblivious Pseudorandom Function)
 - ▶ pseudorandom function $F_k(x)$
 - ▶ OPRF is a protocol with two parties C (input x) and S (input k)
 - ▶ C learns $F_k(x)$ at the end, and nothing else
 - ▶ S learns nothing (in particular, nothing about x)

Example: DH-OPRF

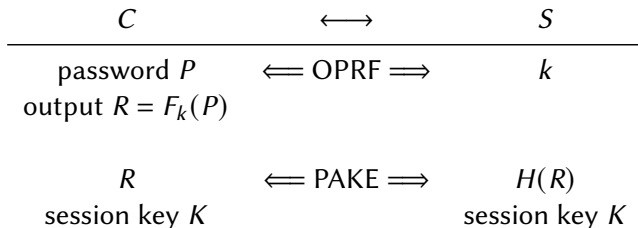
- ▶ l – security parameter
- ▶ group G of prime order q (where $|q| = l$)
- ▶ hash function $H' : \{0, 1\}^l \rightarrow G$, H with range $\{0, 1\}^l$
- ▶ PRF $F : \mathbb{Z}_q \times \{0, 1\}^l \rightarrow \{0, 1\}^l$:

$$F_k(x) = H(x, H'(x)^k)$$

- ▶ protocol:
 1. $C \rightarrow S$: $a = H'(x)^r$, for random $r \in \mathbb{Z}_q$
 2. $S \rightarrow C$: $b = a^k$
 3. C computes $H(x, b^{1/r})$
- ▶ correctness: $b^{1/r} = (H'(x)^r)^{k/r} = H'(x)^k$
- ▶ security: ROM (for hash function) + “one more DH” assumption
 - ▶ informally, after Q oracle queries (oracle returns k -th power) the attacker cannot compute one-more k -th power (moreover, attacker has access to DDH oracle)

Idea: combining OPRF and PAKE

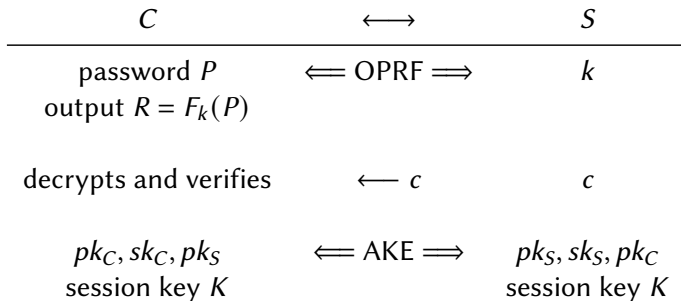
- ▶ S stores $k, H(R)$ for C



- ▶ pre-computation attack is impossible, since R is random to the attacker
- ▶ attacker learns k and $H(R)$ only after S is compromised

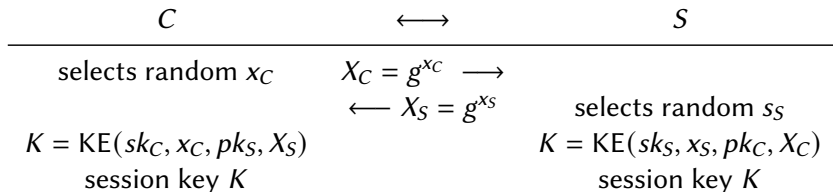
Idea: combining OPRF and AKE

- ▶ assumptions for AKE:
 - ▶ C 's public/private key: pk_C/sk_C
 - ▶ S 's public/private key: pk_S/sk_S
- ▶ AuthEnc – authenticated encryption $c = \text{AuthEnc}_R(pk_C, sk_C, pk_S)$
 - ▶ S stores k, c, pk_C for C



AKE example – HMQV

- ▶ HMQV: variant of DH protocol with implicit authentication of K
- ▶ modifiable for arbitrary finite groups, e.g. elliptic curves
- ▶ multiple variants of MQV (Menezes-Qu-Vanstone) / HMQV (hash MQV)
- ▶ private and public key for participant A: $pk_A = g^{sk_A}$



- ▶ computation:

U :

$$\text{KE}(sk_C, x_C, pk_S, X_S) = H((X_S \cdot pk_S^{e_S})^{x_C + sk_C \cdot e_C}) = H(g^{(x_S + sk_S \cdot e_S)(x_C + e_C \cdot sk_C)})$$

S :

$$\text{KE}(sk_S, x_S, pk_C, X_C) = H((X_C \cdot pk_C^{e_C})^{x_S + sk_S \cdot e_S}) = H(g^{(x_C + e_C \cdot sk_C)(x_S + sk_S \cdot e_S)})$$

- ▶ parameters $e_C = H(X_C, S)$ and $e_S = H(X_S, C)$

Remark – small group confinement

- ▶ DH-like schemes or schemes with security related to DLOG
- ▶ unauthenticated data – group element
- ▶ existence of small subgroups
- ▶ example: DH protocol in (\mathbb{Z}_p^*, \cdot) with generator g
- ▶ let $w \mid (p - 1)$ be a small prime and let $k = (p - 1)/w$
- ▶ attack:
 1. $A \rightarrow E(B): A = g^a$
 2. $E(A) \rightarrow B: A^k$
 3. $B \rightarrow E(A): B = g^b$
 4. $E(B) \rightarrow A: B^k$
- ▶ A and B compute shared secret g^{kab}
- ▶ E can find this secret searching in small subgroup $[g^k]$ (order w)
 - ▶ $(g^k)^w = g^{(p-1)w/w} = g^{p-1} = 1$
- ▶ choose suitable groups and check parameters