Aalto University

Network Security: Diffie-Hellman

Tuomas Aura, Aalto University

CS-E4300 Network security

Outline

1. Discrete logarithm problem
2. Diffie-Hellman key exchange
3. Impersonation and MitM
4. Authenticated DH
5. Misbinding
6. A more realistic protocol
7. Perfect forward secrecy (PFS)

Diffie-Hellman overlaps with Information Security and any basic cryptography course. It is covered in detail here to ensure that all students have sufficient understanding of DH and the impersonation attack.

Modulo arithmetic

Please refer to cryptography literature for the details

- Exponentiation in multiplicative group Z_p^*:
  - Choose a large prime number p (e.g. 2048 bits long)
  - Z_p^* is the group of integers 1..p-1;
    group operation is multiplication modulo p
  - Exponentiation x^k means multiplying x with itself k times modulo p
  - g is a generator if g^k for k=0,1,2,3,… produces all the values 1..p-1
- For Diffie-Hellman, choose parameters p and g
  - Many critical details; see crypto literature!
- Exponentiation is commutative: $(g^x)^y = (g^y)^x$
  i.e. $(g^x \bmod p)^y \bmod p = (g^y \bmod p)^x \bmod p$

Elliptic curve (EC)

Please refer to cryptography literature for the details

- Points on an elliptic curve form an additive group
  - Commonly used curves: Curve25519, Curve448
  - See cryptography literature for details
- Point multiplication $n \cdot P$ means adding P to itself n times
  - n is an integer; P is a point on the elliptic curve
- Point G is a generator point if $k \cdot G$ for k=0,1,2,3,… produces all the values of the group or a large subgroup
- Point multiplication is commutative: $n \cdot m \cdot G = m \cdot n \cdot G$

Discrete logarithm problem

- Discrete logarithm problem in Z_p^*: given $g^k \bmod p$, solve k
  - Believed to be a hard problem for large primes p and random k
  - Typical p 1024..8096 bits; k 256 bits
- Discrete logarithm problem in EC: given $n \cdot P$, solve n
  - Believed to be a hard problem
  - Typical point lengths are 160..571 bits, depending on the curve; multiplier n 256 bits
  - Why EC? Shorter key lengths and lower computation cost for the same level of security

Unauthenticated Diffie-Hellman in Z_p^*

- A and B have previously agreed on g and p
- All operations are in Z_p^* i.e. modulo p

A chooses a random x and computes key share g^x
B chooses a random y and computes key share g^y

1. A → B: A, g^x
2. B → A: B, g^y

A calculates shared secret $K = (g^y)^x$

B calculates shared secret $K = (g^x)^y$

- It works because exponentiation is commutative
- Sniffer learns g^x and g^y ; cannot compute x, y, or g^{xy}

Elliptic Curve Diffie-Hellman (ECDH)

- A and B have previously agreed on a curve and G

A chooses a random d_A and computes key share $Q_A = d_A \cdot G$

B chooses a random d_B and computes key share $Q_B = d_B \cdot G$

1. A → B: A, Q_A
2. B → A: B, Q_B

A computes the shared secret $SK = d_A \cdot Q_B = d_A \cdot d_B \cdot G$

B computes the shared secret $SK = d_B \cdot Q_A = d_B \cdot d_A \cdot G$

For protocol designers, DH and ECDH are interchangeable algorithms

- It works because point multiplication is commutative
- Sniffer learns Q_A and Q_B ; cannot compute d_A, d_B, or SK

Diffie-Hellman assumption

- Diffie-Hellman assumption in Z_p^*:

  given g^x and g^y, hard to solve $K = g^{xy}$

- Diffie-Hellman assumption in EC:

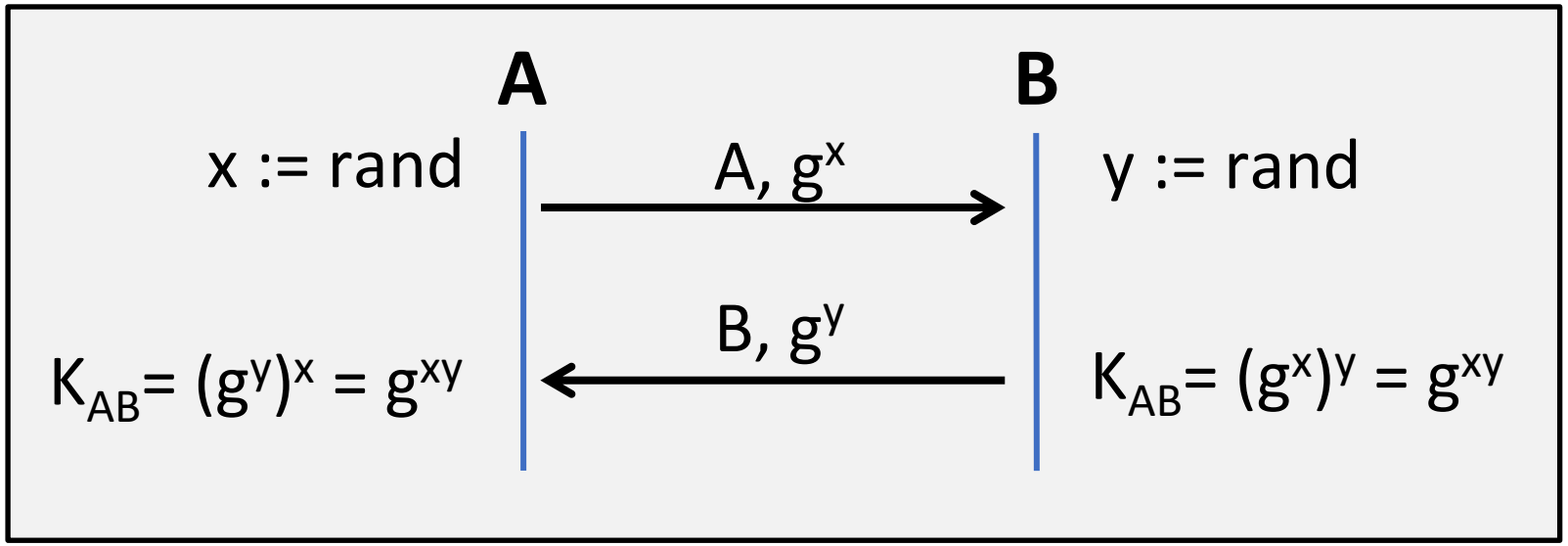
  given $d_A \cdot G$ and $d_B \cdot G$, hard to solve $K = d_A \cdot d_B \cdot G$

- Believed to be as hard as the discrete logarithm problem
  - Ability to compute discrete logarithms also breaks the DH assumption
  - Quantum computers could compute discrete logarithms

Domain parameters

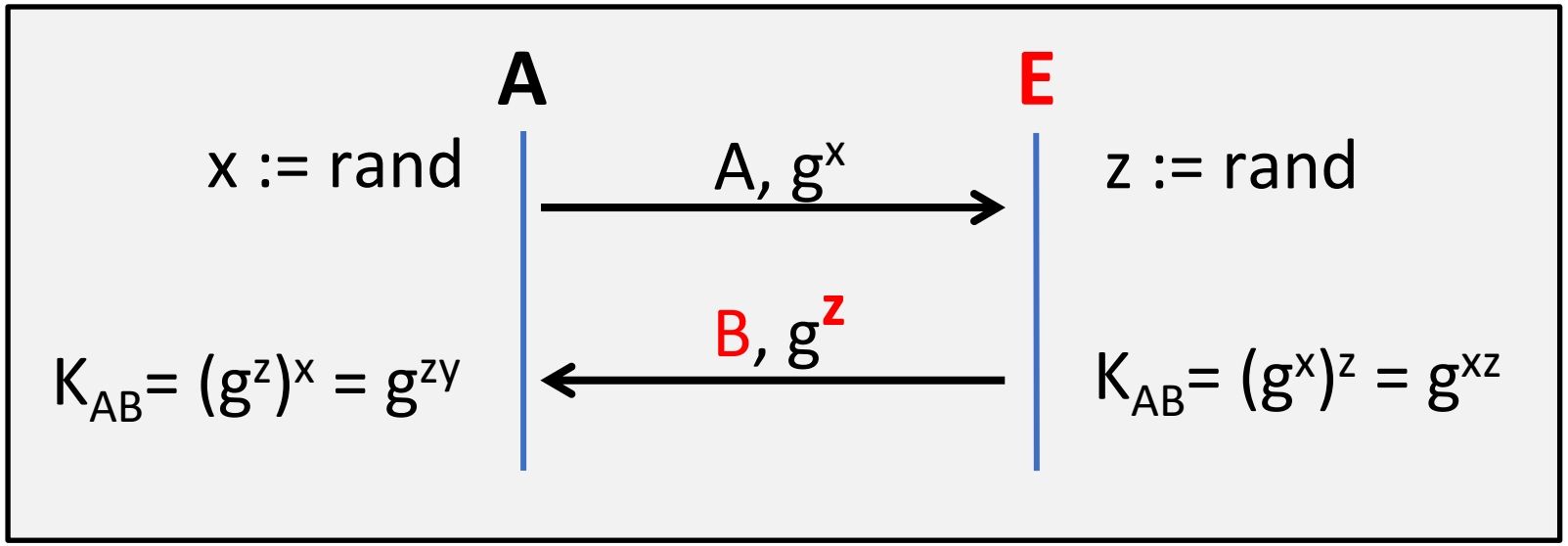
- Domain parameters in Diffie-Hellman:
  - In Z_p^*, A and B must agree on the prime p and generator g
  - In ECDH, A and B must agree the curve and generator point G
- How to agree on the domain parameters?
  - Method 1: standardized parameters for each protocol or application
  - Method 2: one party chooses and signs the parameters
  - Method 3: negotiation where one party offers parameters, and the other party chooses from them
- Protocol standards usually allow many key lengths or ECDH curves, and the key-exchange starts with parameter negotiation

Sniffing



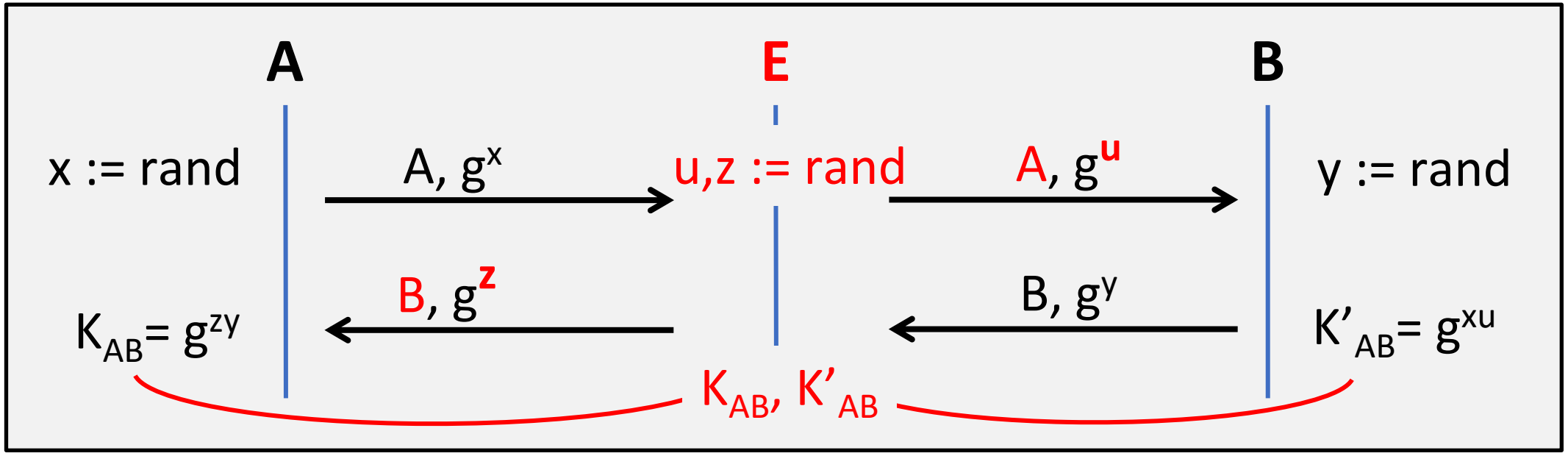
- Unauthenticated Diffie-Hellman is secure against passive attackers
  - Not possible to discover the shared secret K_{AB} by sniffing the key shares

Impersonation attack



- Unauthenticated Diffie-Hellman is vulnerable to an active attacks such as impersonation:
  - Shared secret key was created, but with whom?

Man in the Middle (MitM)



- Attacker impersonates A to B, and B to A
- Attacker creates shared session keys with both A and B
- Later, attacker can forward data between the two “secure” sessions

Authenticated DH

1. A → B: A, g^x, $S_A(g^x)$, $Cert_A$
2. B → A: B, g^y, $S_B(g^y)$, $Cert_B$

SK = $h(g^{xy})$

Note: This is still an impractical toy protocol. Please read further

- $S_A(g^x)$ = A's signature
- $Cert_A$ = standard (X.509) public-key certificate or certificate chain
  - Subject name in the certificate must be A
  - B verifies the signature with A's public key from the certificate
- $h(g^{xy})$ = key material for deriving all necessary session keys
- Authentication prevents impersonation and MitM attacks

Authenticated DH with key confirmation

- Three messages needed for authentication and key confirmation

1. A → B: A, B, N_A, g^x
2. B → A: A, B, N_B, g^y, S_B("Msg2", N_A, N_B, g^x , g^y), $Cert_B$,
3. A → B: A, B, S_A("Msg3", N_A, N_B, g^x , g^y), $Cert_A$

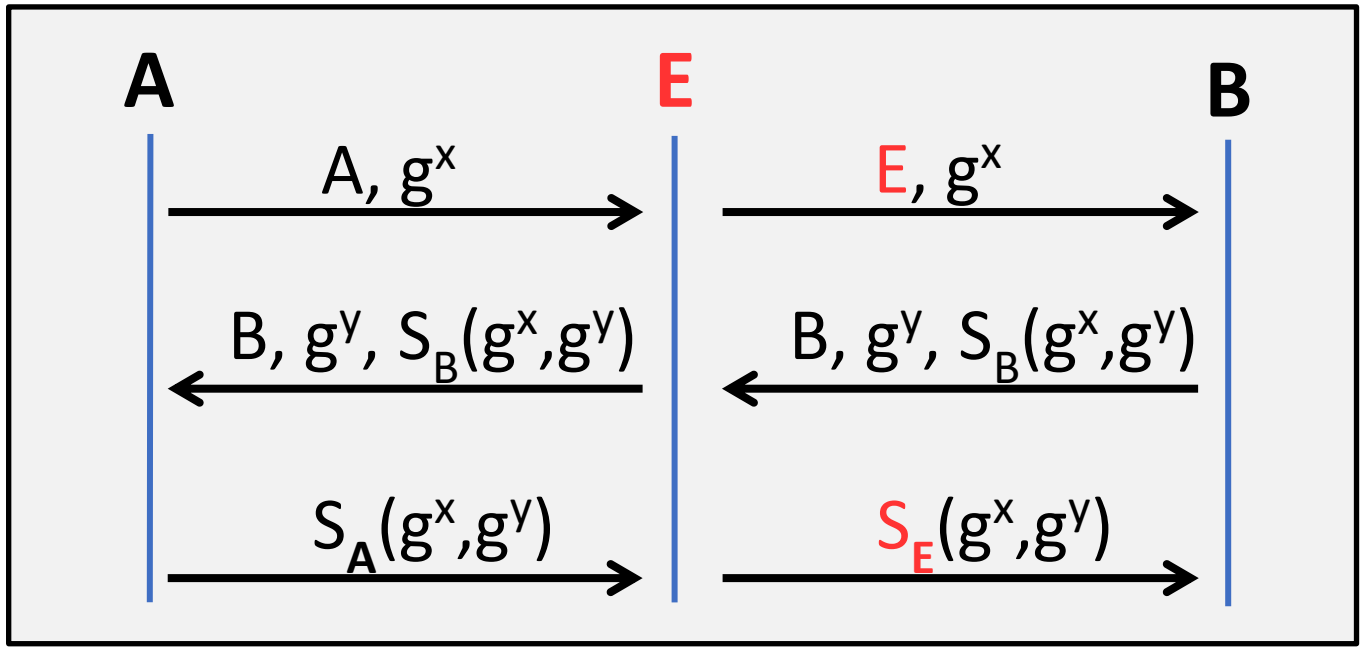
SK = h(N_A, N_B, g^{xy})

Still not a good protocol! Please read further

- Signatures on fresh data authenticate the endpoints
- Key confirmation: signatures prove that each endpoint knows all the parameters needed to compute the session key
  - Endpoints must trust each other about knowing the exponent

Misbinding attack

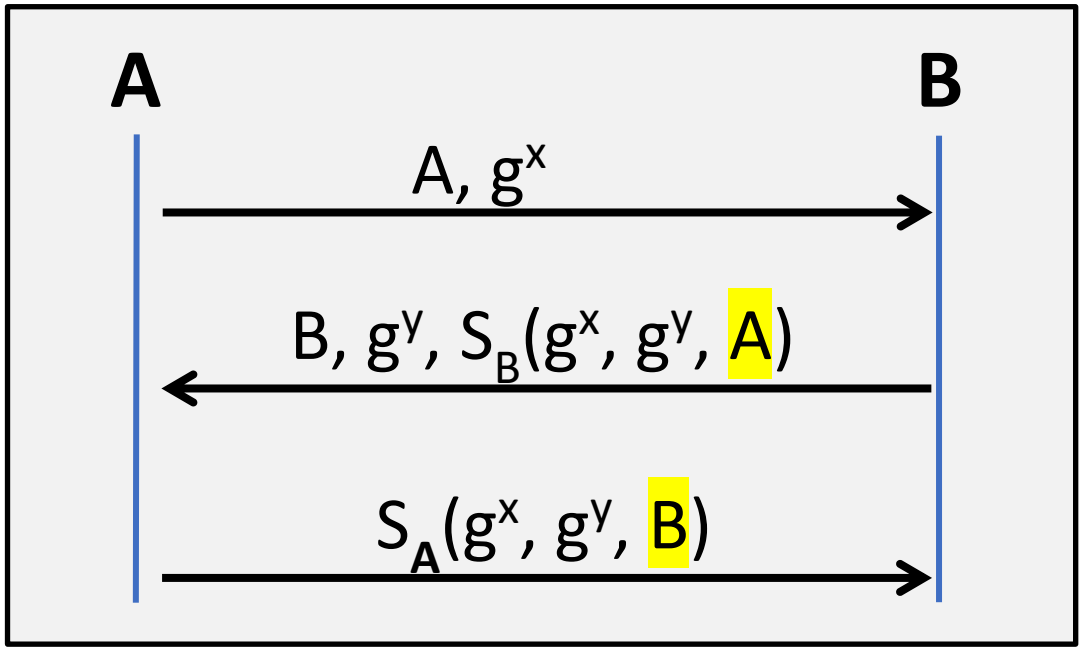
- Misbinding of the initiator: B thinks it is connected to E. In fact, A and B are connected



- E is a dishonest insider (E can legitimately connect to B)
- Misbinding of the responder is similarly possible

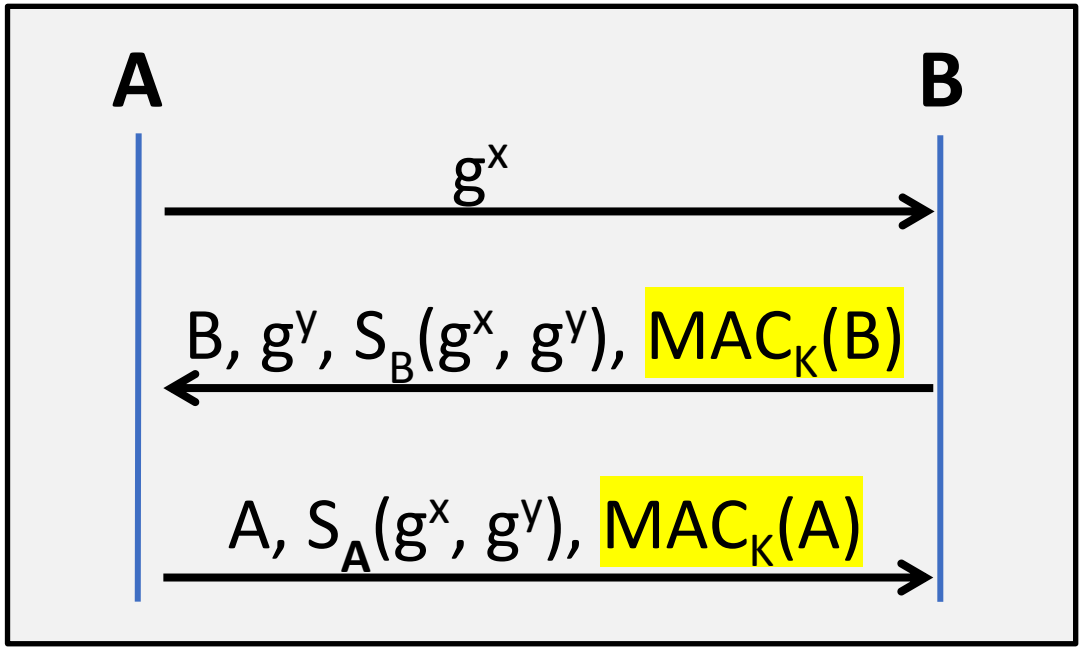
Solutions to misbinding:
check peer identifier

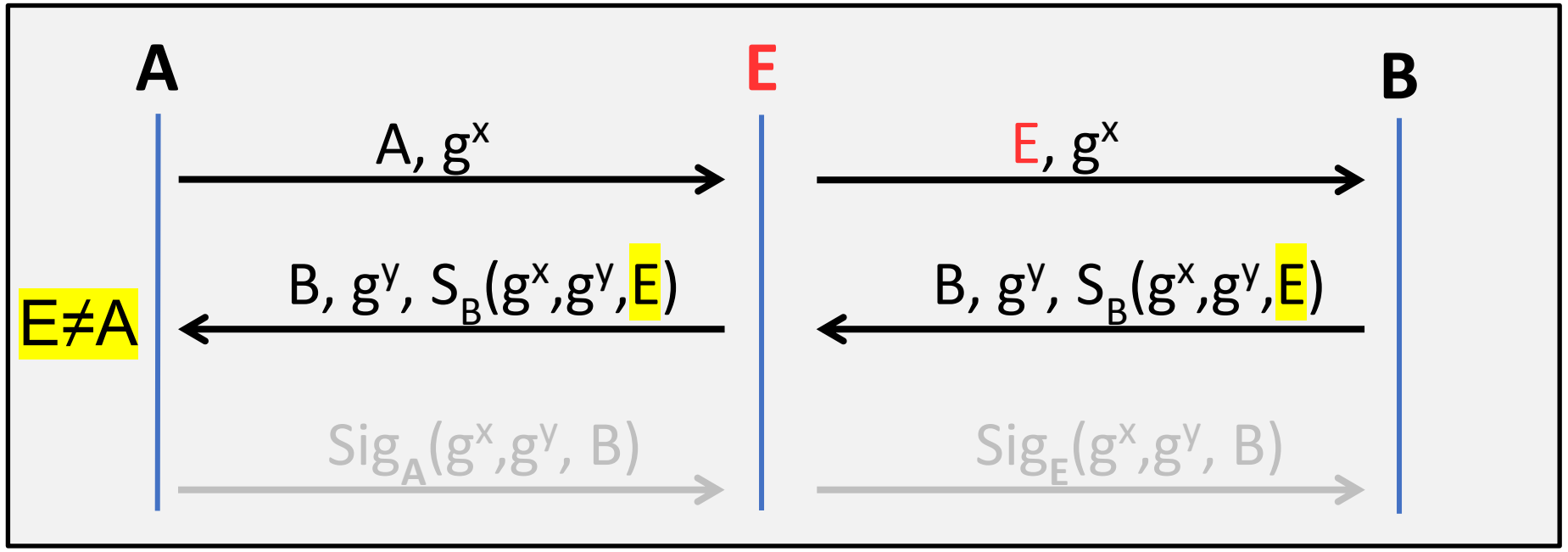
ISO 9798-3



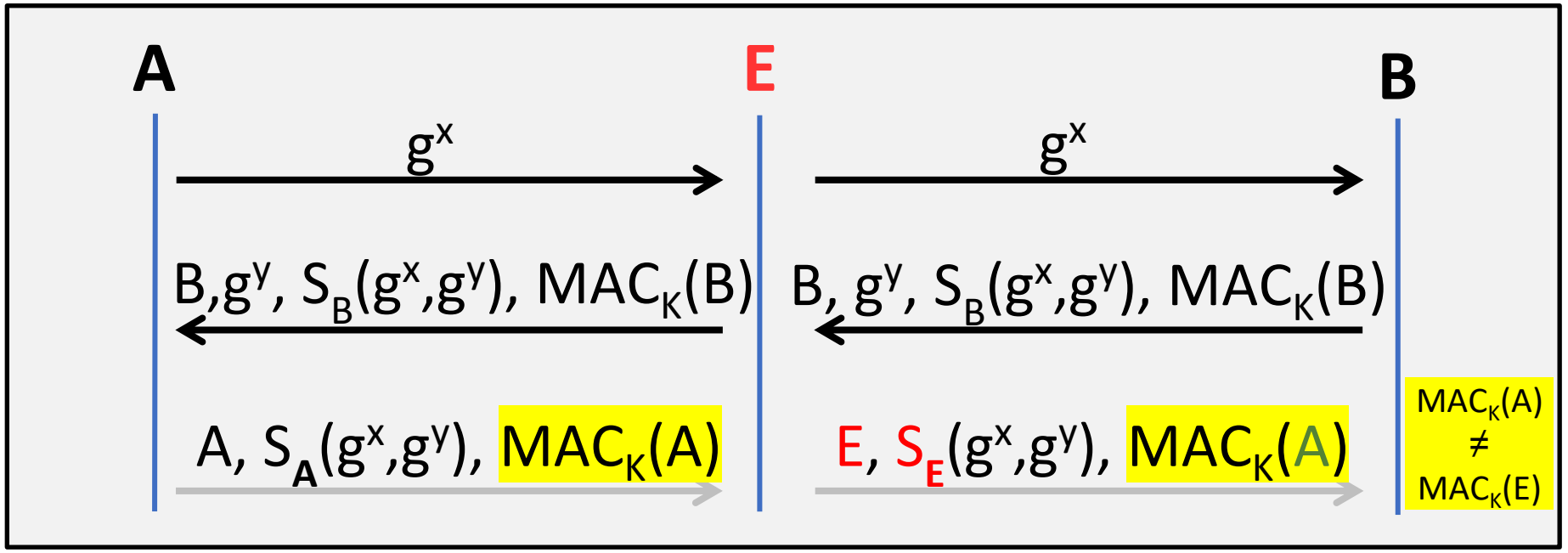
SIGMA

(easier security proofs, and slightly better protection in case of an incompetent CA)





Detecting misbinding of initiator in ISO 9798-3



Detecting misbinding of initiator in SIGMA

A MORE REALISTIC AUTHENTICATED DIFFIE-HELLMAN PROTOCOL

Authenticated DH

- Signed Diffie-Hellman with nonces and key confirmation:

1. $A \rightarrow B$: A, B, N_A, g, p, g^x, S_A("Msg1", A, B, N_A, g, p, g^x), $Cert_A$
2. $B \rightarrow A$: A, B, N_B, g^y, S_B("Msg2", A, B, N_B, g^y), $Cert_B$,
   MAC_{SK}(A, B, "Responder done.")
3. $A \rightarrow B$: A, B, MAC_{SK}(A, B, "Initiator done.")

$SK = h(N_A, N_B, g^{xy})$

- Prevents impersonation, MitM and misbinding attacks
- Why so complicated?

Authenticated DH

- Signed Diffie-Hellman with nonces and key confirmation:

1. A → B: A, B, N_A, g, p, g^x, S_A("Msg1", A, B, N_A, g, p, g^x), $Cert_A$
2. B → A: A, B, N_B, g^y, S_B("Msg2", A, B, N_B, g^y), $Cert_B$, MAC_{SK}(A, B, "Responder done.")
3. A → B: A, B, MAC_{SK}(A, B, "Initiator done.")

SK = h(N_A, N_B, g^{xy})

- Signatures and certificates for authentication, nonces for freshness, MAC for key confirmation

Authenticated DH

- Signed Diffie-Hellman with nonces and key confirmation:

1. A → B: A, B, N_A, g, p, g^x, S_A("Msg1", A, B, N_A, g, p, g^x), $Cert_A$
2. B → A: A, B, N_B, g^y, S_B("Msg2", A, B, N_B, g^y), $Cert_B$, MAC_{SK}(A, B, "Responder done.")
3. A → B: A, B, MAC_{SK}(A, B, "Initiator done.")

SK = h(N_A, N_B, g^{xy})

- Signatures and certificates for authentication, nonces for freshness, MAC for key confirmation

Authenticated DH

- Signed Diffie-Hellman with nonces and key confirmation:

1. A → B: A, B, N_A, g, p, g^x, S_A("Msg1", A, B, N_A, g, p, g^x), $Cert_A$
2. B → A: A, B, N_B, g^y, S_B("Msg2", A, B, N_B, g^y), $Cert_B$, MAC_{SK}(A, B, "Responder done.")
3. A → B: A, B, MAC_{SK}(A, B, "Initiator done.")

SK = h(N_A, N_B, g^{xy})

- Signatures and certificates for authentication, nonces for freshness, MAC for key confirmation

Authenticated DH

- Signed Diffie-Hellman with nonces and key confirmation:

1. $A \rightarrow B$: A, B, N_A, g, p, g^x, S_A("Msg1", A, B, N_A, g, p, g^x), $Cert_A$
2. $B \rightarrow A$: A, B, N_B, g^y, S_B("Msg2", A, B, N_B, g^y), $Cert_B$, MAC_{SK}(A, B, "Responder done.")
3. $A \rightarrow B$: A, B, MAC_{SK}(A, B, "Initiator done.")

$SK = h(N_A, N_B, g^{xy})$

- Signatures and certificates for authentication, nonces for freshness, MAC for key confirmation

Authenticated DH

- Signed Diffie-Hellman with nonces and key confirmation:

1. $A \rightarrow B$: A, B, N_A, g, p, g^x, S_A("Msg1", A, B, N_A, g, p, g^x), $Cert_A$
2. $B \rightarrow A$: A, B, N_B, g^y, S_B("Msg2", A, B, N_B, g^y), $Cert_B$, MAC_{SK}(A, B, "Responder done.")
3. $A \rightarrow B$: A, B, MAC_{SK}(A, B, "Initiator done.")

$SK = h(N_A, N_B, g^{xy})$

- Signatures and certificates for authentication, nonces for freshness, MAC for key confirmation

Ephemeral Diffie-Hellman (DHE)

- Perfect forward secrecy (PFS): session keys and data from past sessions are safe even if the long-term secrets, such as private keys, are later compromised !
  - Even participants themselves cannot recover old session keys
- Ephemeral DH (DHE): new random DH exponents for every key exchange, forget the exponent values afterwards → PFS
  - Similarly, ephemeral ECDH (ECDHE)
  - Cost-security trade-off: replace DH exponents periodically, e.g. once in a day or an hour, and use nonces for freshness: $SK = h(N_A, N_B, g^{xy})$

Diffie-Hellman and nonces

- Are the nonces needed in Diffie-Hellman?

1. A → B: A, B, N_A, g, p, g^x, S_A("Msg1", A, B, N_A, g, p, g^x), $Cert_A$
2. B → A: A, B, N_B, g^y, S_B("Msg2", A, B, N_B, g^y), $Cert_B$, MAC_{SK}(A, B, "Responder done.")
3. A → B: A, B, MAC_{SK}(A, B, "Initiator done.")

SK = h(N_A, N_B, g^{xy})

- Old DH implementations reuse exponents
  → Saving on computation. Lack of PFS. Nonces needed for freshness
- After Snowden, PFS has become mandatory → Ephemeral DH. Nonces optional
- Prudent protocol design still separates the two concerns: nonces for freshness of authentication and session key, DH for secrecy and new exponents for PSF