Key Establishment

Colin Boyd

Department of Telematics
NTNU

May 2014

NTNU – Trondheim
Norwegian University of
Science and Technology

Outline

Designing a Protocol

We choose a common scenario. There are a wide variety of alternatives as discussed later.

- A set of users, any two of whom may participate in the protocol.
- An entity called the *server* which will also engage in the protocol.
- All users trust the server to execute the protocol faithfully and not to engage in any other activity which will deliberately compromise their security.

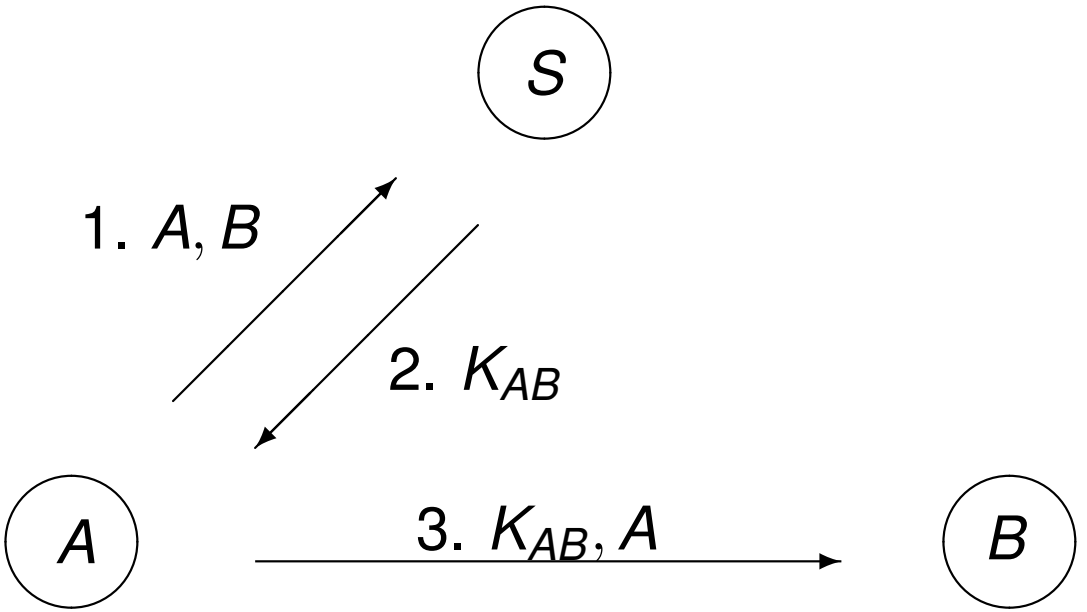
Our protocol involves three entities or *principals*.

- Two users whom we denote A and B (often expanded to *Alice* and *Bob*)
- The trusted server S.

The protocol is used to establish a new secret key K_{AB} to be used for subsequent secure communication. The role of S is to generate K_{AB} and transport it to A and B. The aims of the protocol are as follows.

- At the end of the protocol A and B should both know the value of K_{AB}.
- A and B should both know the identities of all other principals who know K_{AB}.
- A and B should know that K_{AB} is newly generated.

Protocol 1



The protocol consists of three messages. User A contacts S by sending the identities of the two parties who are going to share the new session key; S returns the key K_{AB} to A; A passes K_{AB} on to B.

Confidentiality

The problem with Protocol 1 is that the session key K_{AB} must be transported to A and B but to *no other entities*.

Security Assumption

The adversary is able to eavesdrop on all messages sent in a cryptographic protocol.

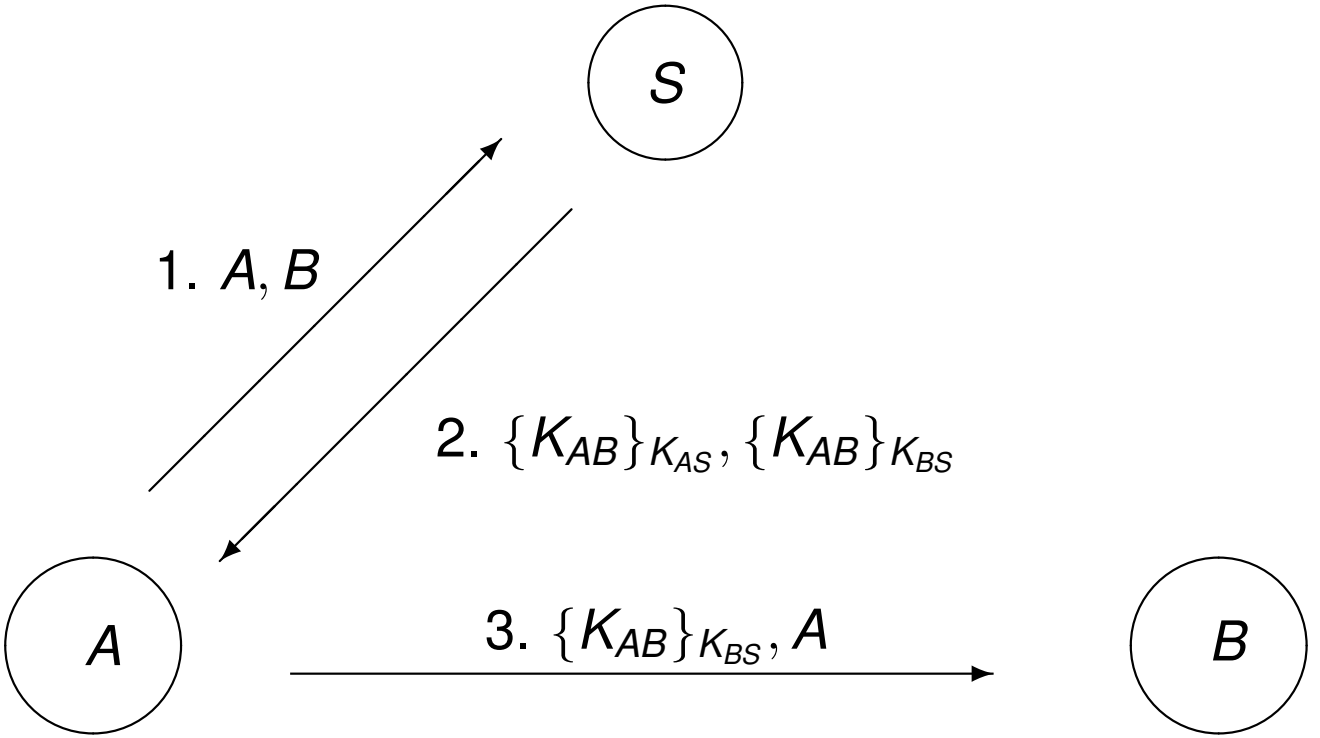
The need to keep K_{AB} confidential leads to Protocol 2. We will make the normal assumption that the server S initially shares a long-term secret key with each user of the system.

- K_{AS} is shared between S and A.
- K_{BS} is shared between S and B.
- K_{AB} is the new session key generated by S.
- $\{X\}_K$ denotes encryption of X using key K.

Cryptographic algorithms

- It is common when considering cryptographic protocols to ignore the details of the cryptographic algorithms involved.
- When we come to look at formal models we need to know precisely what encryption properties are provided.
- For now assume that a passive eavesdropper will gain nothing because K_{AB} may only be read by the legitimate recipients, since only they have the keys required to decrypt.

Protocol 2



The session key is now protected during its transmission to each party.

Authentication

- Protocol 2 is just as insecure as Protocol 1 but for a completely different reason.
- Protocol 2 lacks information about who else has the key is not protected. The adversary is not able just to eavesdrop on messages sent; he can also capture messages and alter them.

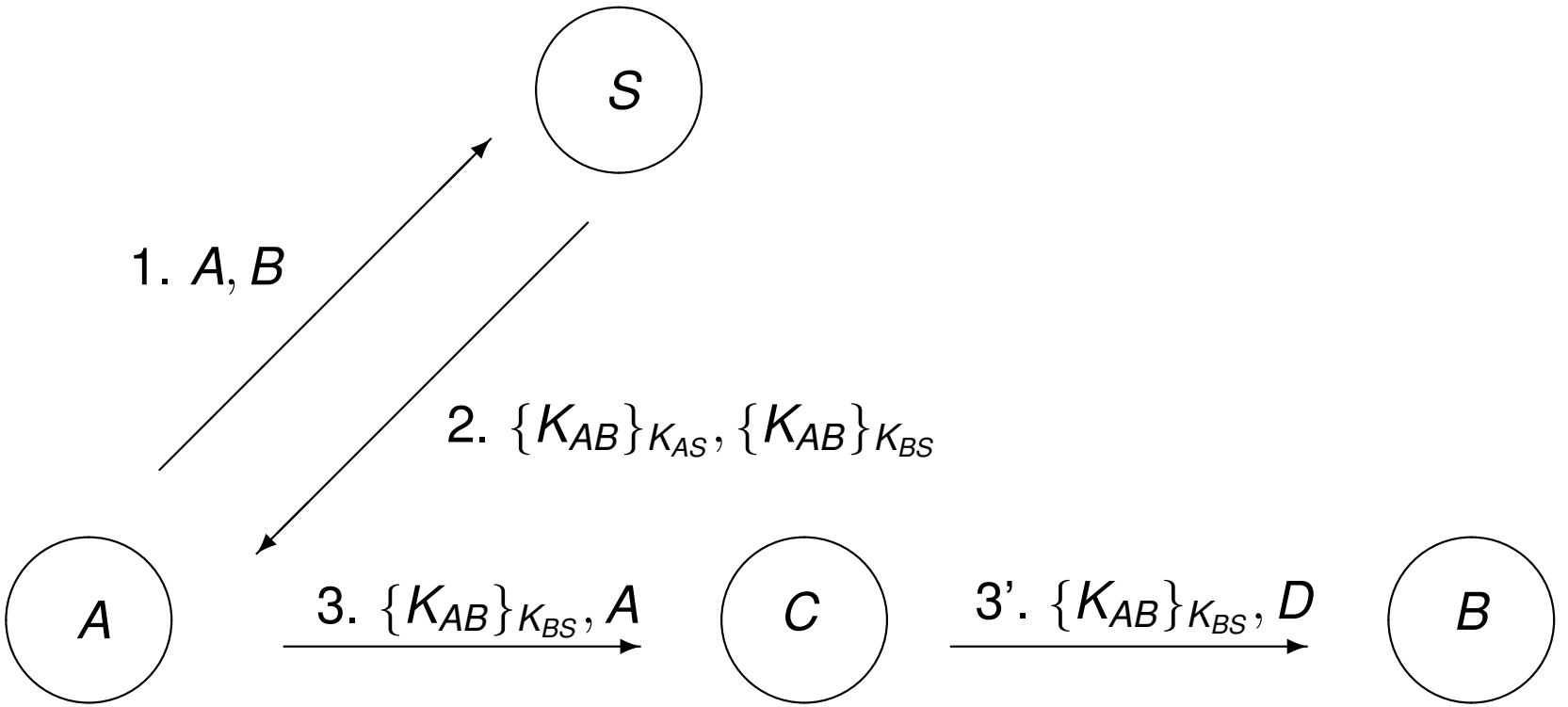
Security Assumption

The adversary is able to alter all messages sent in a cryptographic protocol using any information available to him. In addition the adversary can re-route any message to any other principal. This includes the ability to generate and insert completely new messages.

The reason for the difficulty in designing authentication protocols now begins to clarify.

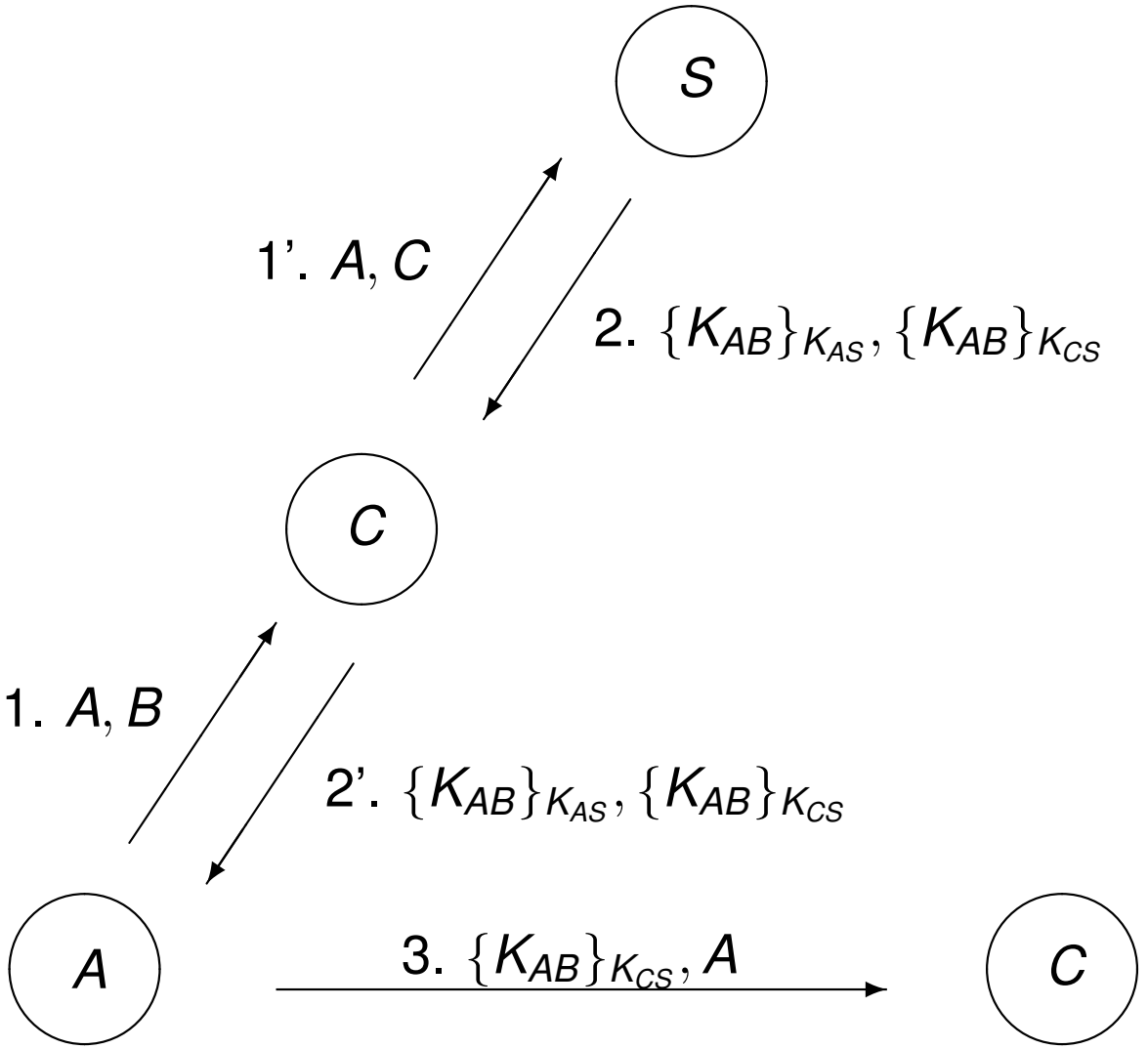
- In distinction to ordinary communications protocols there is an unknown and unpredictable malicious principal involved.
- Although there are very few messages involved in a legitimate run of the protocol, there are an infinite number of variations in which the adversary participates.
- Each variation must satisfy the protocol's security requirements.

Attack on Protocol 2



- Adversary C intercepts the message from A to B and substitutes C's name for A's (where C could be any identity including C's own). The consequence is that B believes that he is sharing the key with C whereas he is in fact sharing it with A.
- The subsequent results may include such actions as B giving away information to A which should only have been shared with C.
- Notice that C does not obtain K_{AB} but we still regard the protocol as broken since it does not satisfy our requirement that the users should know who else knows the session key.

Alternative attack on Protocol 2

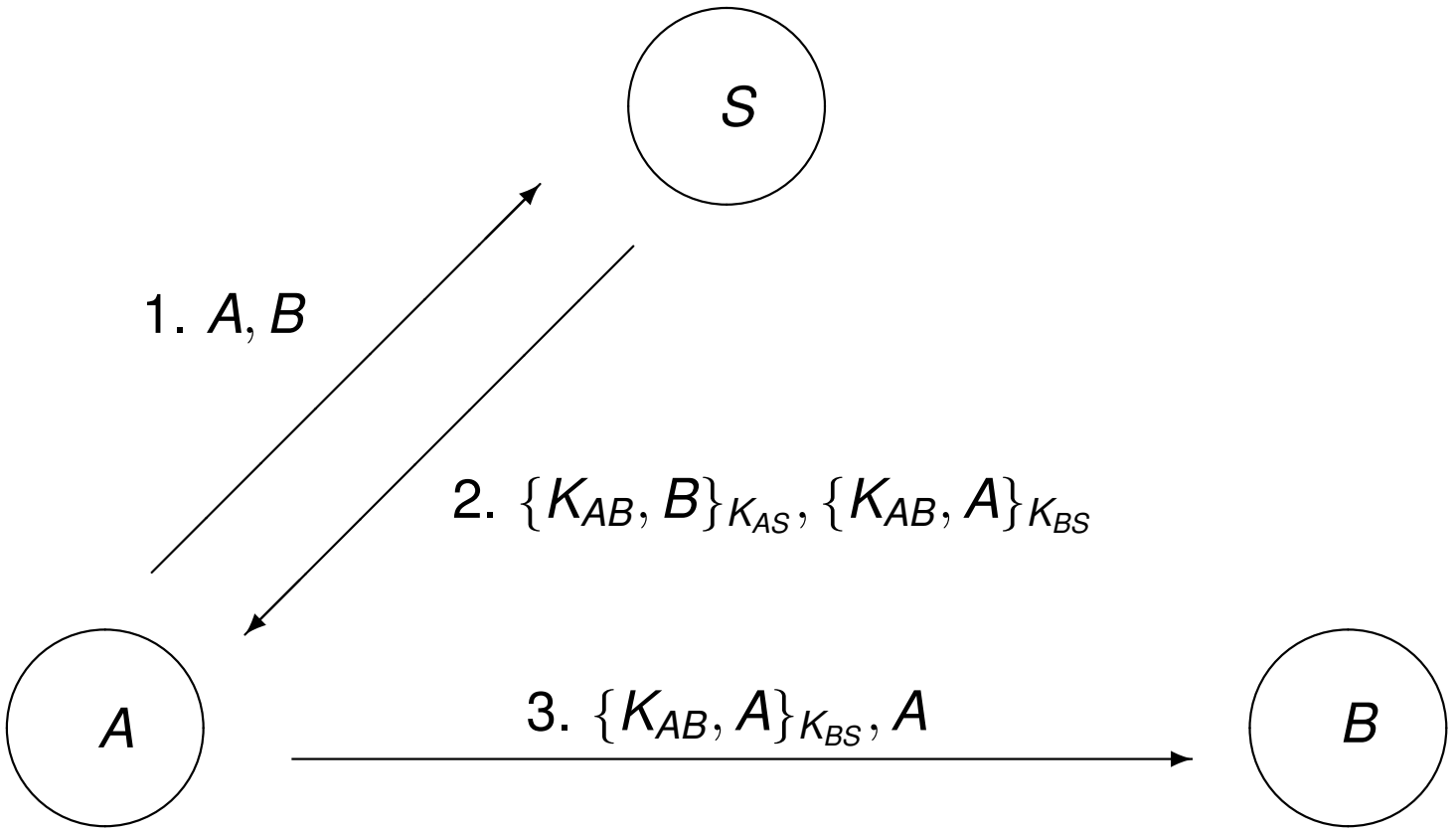


- C alters the message from A to S so that S encrypts the key K_{AB} with C's key K_{CS} instead of B's key.
- Since A cannot distinguish between encrypted messages meant for other principals she will not detect the alteration.
- C simply collects the message from A intended for B so that B will not detect any anomaly.

The result of this attack is that A will believe that the protocol has been successfully completed with B whereas in fact C knows K_{AB} and so can masquerade as B as well as learn all the information that A intends to send to B.

- In contrast to the previous attack this one will only succeed if C is a legitimate user known to S. This is a realistic assumption - it is well documented that insiders are often more of a threat than outsiders.
- To overcome the attack the names of the users need to be bound cryptographically to the value of K_{AB}. This leads to Protocol 3 where the names of A and B are included in the encrypted messages received from S.
- The encryption algorithm used by S must prevent the possiblity to alter the value of the encrypted messages. Thus *authenticated encryption* must be used.

Protocol 3



Replay

The next problem stems from the difference in quality between the *long-term keys* shared initially with S, and the *session key* K_{AB} generated for each protocol run.

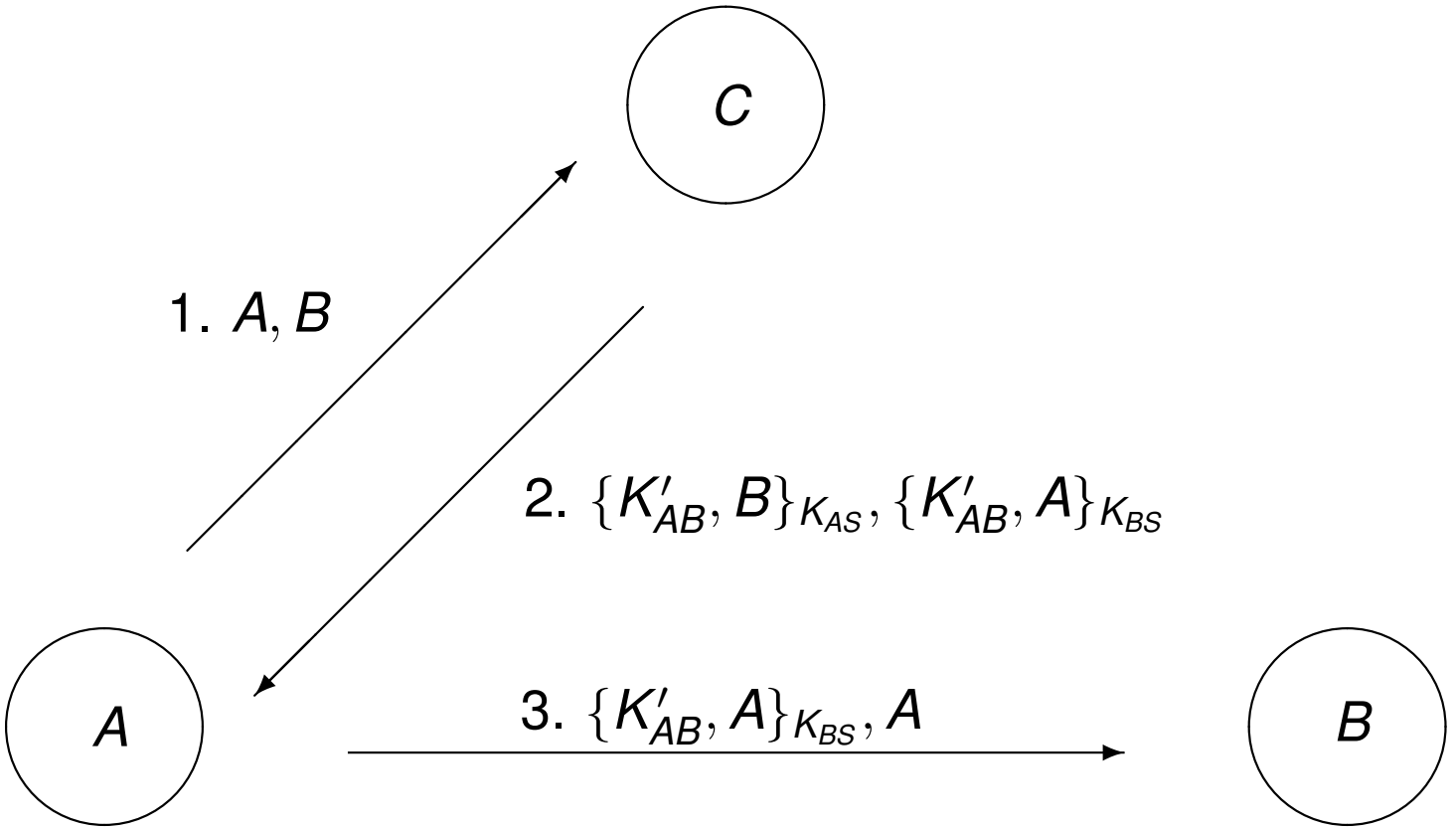
- A new key is required for each session.
- Session keys are expected to be vulnerable to attack.
- Session keys will be likely to be used with a variety of data of regular formats, making them targets for cryptanalysis.
- Session keys may be discarded carelessly after the session is closed.

- A whole class of attacks is based on the notion that old keys may be *replayed* in a subsequent session.
- Notice that even if *A* is careful in the management of session keys used by her, compromise of a session key by *B* may still allow replay attacks when *A* communicates with *B*.

Security Assumption

An adversary is able to obtain the value of the session key K_{AB} used in any (sufficiently old) previous run of the protocol.

Attack on Protocol 3



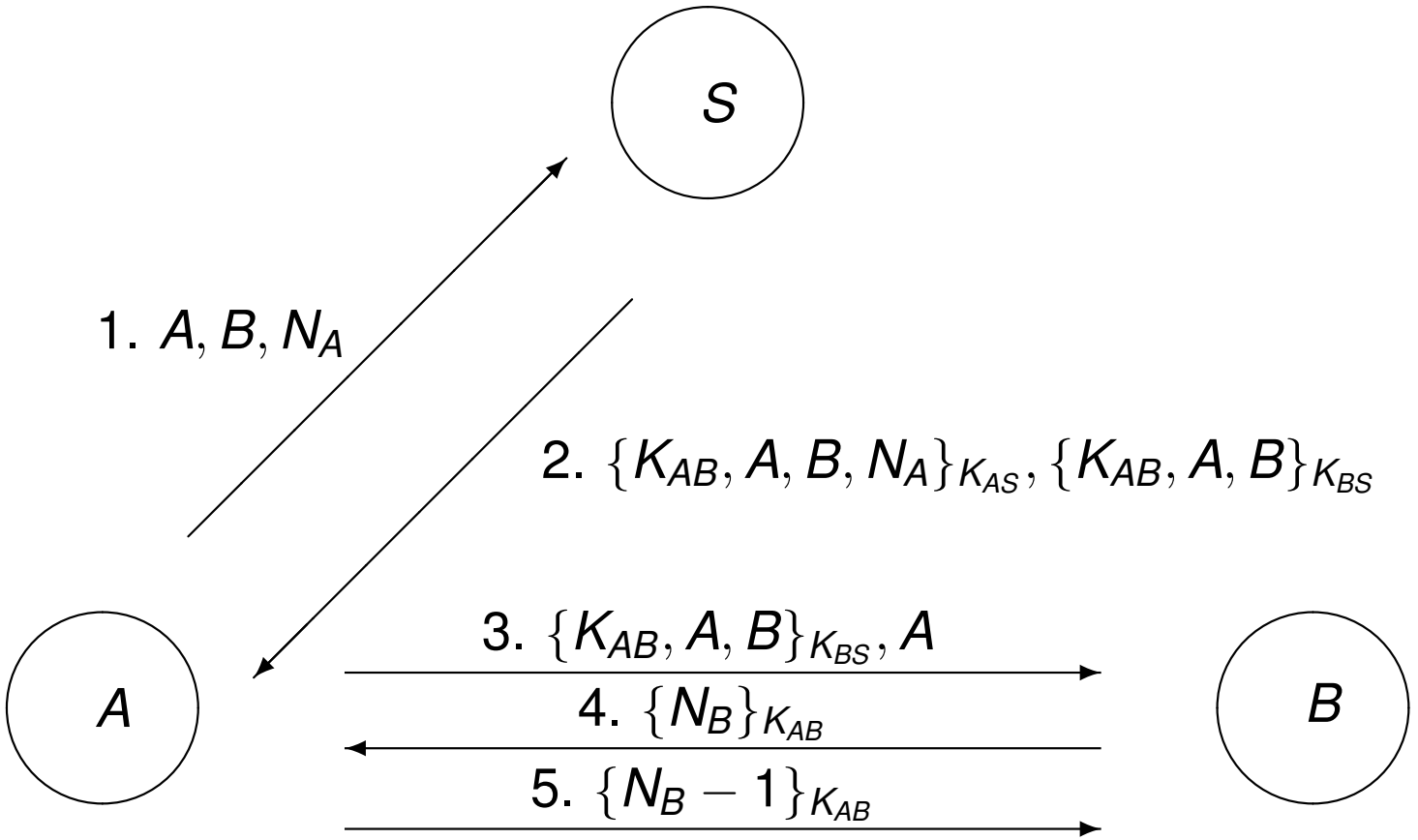
The key K'_{AB} is an old session key used by A and B in a previous session; C can be expected to know the value of K'_{AB}.

Avoiding replay

- There are various mechanisms that may be employed to allow users to check that session keys have not been replayed. We will improve our protocol using *challenge-response*.
- A will generate a new random value N_A at the start of the protocol and send this to S with the request for a new key.
- If N_A is subsequently received by A bound together with the session key, then A can deduce that the key has not been replayed.

It is not convenient for B to send his own nonce to S. A possible solution is that A can pass on her assurance to B; once A is satisfied that the key is new then she can respond to a nonce N_B generated by B and sent by B protected by K_{AB} itself.

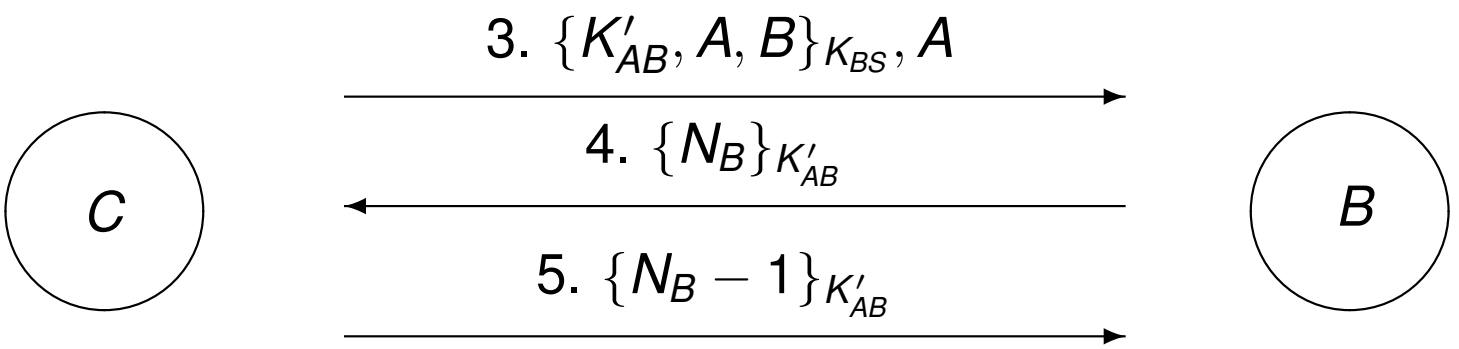
Protocol 4 (Needham–Schroeder)



Protocol 4 is essentially the same as one of the most celebrated in the subject of key establishment protocols.

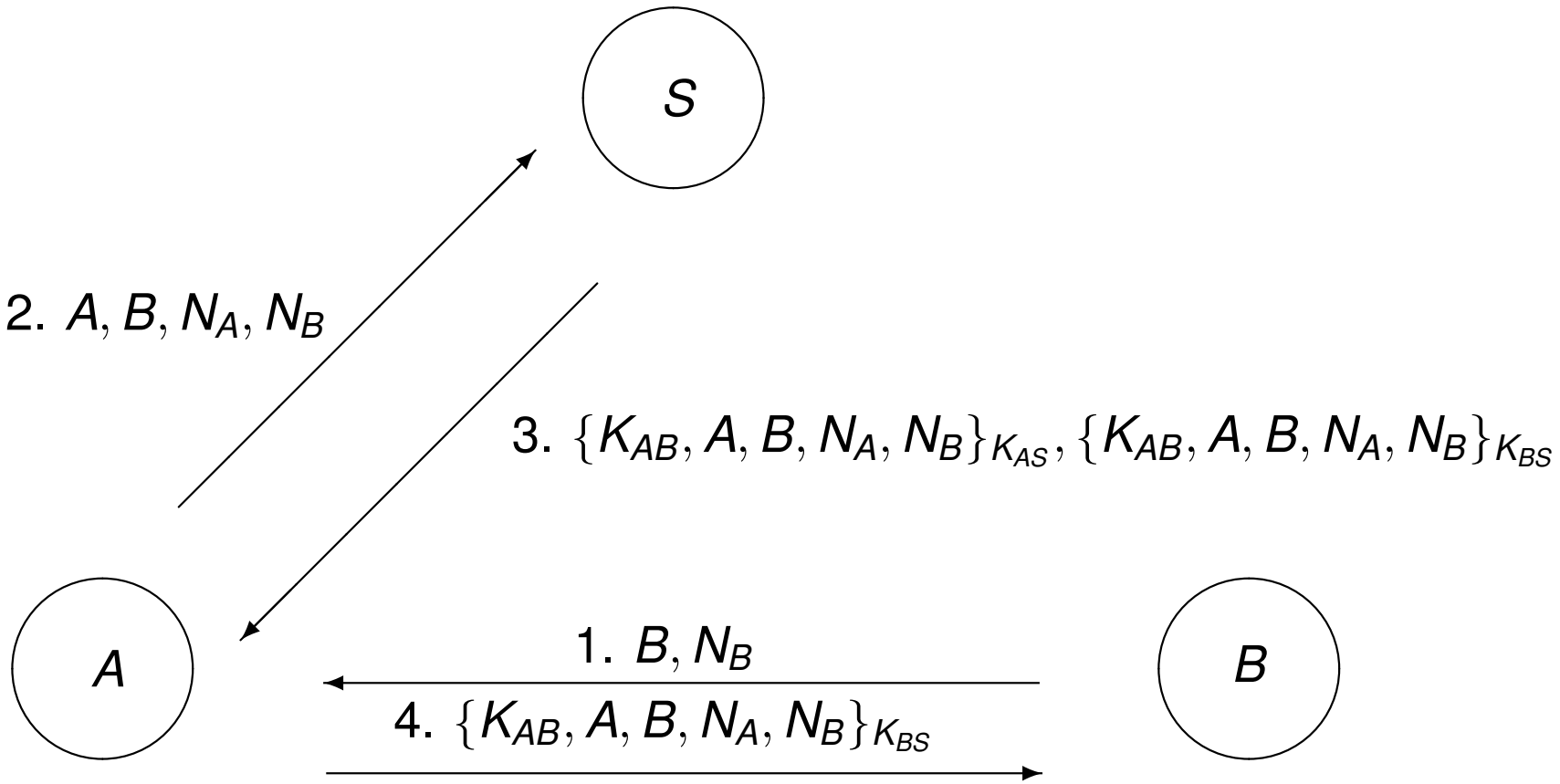
- This is the Needham and Schroeder protocol published in 1978.
- It has been the basis for a whole class of related protocols.
- Unfortunately it is vulnerable to an attack due to Denning and Sacco in 1981.
- The assumption that A will act properly in assuring B that the key is new is unfounded. Since A is any user, and could be the adversary C, this assumption is unrealistic.

Denning–Sacco attack on Protocol 4



- Once the attack has been spotted, it is relatively easy to suggest ways of overcoming it. We dispense with the assumption that it is inconvenient for B to send his challenge to S as well as A.
- Protocol 5 is secure if the cryptographic algorithm used has the properties of both confidentiality and integrity, and the server S acts correctly.

Protocol 5


S
2. A, B, N_A, N_B
3. $\{K_{AB}, A, B, N_A, N_B\}_{K_{AS}}, \{K_{AB}, A, B, N_A, N_B\}_{K_{BS}}$
A
1. B, N_B
4. $\{K_{AB}, A, B, N_A, N_B\}_{K_{BS}}$
B

- In Protocol 5 both parties send their own nonce to S who returns it bound to the key.
- Protocol 5 can be proven secure (next lecture).
- Notice that in Protocol 4 A could verify both
  - the key is new and known only by A, B and S
  - B has in fact received the key.
- In Protocol 5 neither A nor B can deduce at the end of a successful protocol run that the other has actually received K_{AB}.

Outline

Protocol architectures

In the sample protocol above, the key was generated by a trusted server. In other protocols one or more of the users may generate the key. It is common to consider two cases.

Key Transport Protocols Here the key is generated by one of the principals and transferred to the other(s).

Key Agreement The principals both (all) contribute information which together establishes the key.

- Key transport is commonly associated with symmetric-key protocols with an online server.
- Key agreement is commonly associated with asymmetric-key protocols without a server.
- None of these situations is necessary.

Basic properties for key establishment

- Basic goals:
  - parties should accept the same key if adversary is benign;
  - adversary cannot obtain the session key (often called *implicit key authentication*).
- Adversary capabilities:
  - controls the network;
  - reveal session keys from other (non-target) sessions;
  - corrupt parties not from other (non-target) sessions to obtain long-term keys.

Diffie–Hellman key agreement

- Diffie–Hellman key agreement was published in 1976.
- Computations takes place in a suitable group with element g of large order.
- Principals A and B choose random values r_A and r_B and exchange values g^{r_A} and g^{r_B}.
- The shared secret is $Z_{AB} = g^{r_A r_B}$.
- The unauthenticated values r_A and g^{r_A} are known as *ephemeral* private and public keys.
- The session key K_{AB} is formed from the shared secret using a *key derivation function*.

Key derivation functions

- Many sources do not specify the key derivation function. The IEEE P1363 standard as well as NIST SP-800A (2007) specifies the session key as

$$K_{AB} = H(Z_{AB}, \text{Info})$$

  where H is SHA-1 and Info is a set of optional data including an algorithm identifier for the session key, a counter, public information of A or B, and any other shared data.
- In formal models H is often modelled as a random oracle.
- Standard model proofs may require H to be specified using a randomness extractor.

Advanced properties for key agreement

Forward secrecy Compromise of long term keys of both parties *after session is complete* should not compromise session key.

Key compromise impersonation (KCI) resistance The adversary should not be able to masquerade to A as any different party B.

Leakage of ephemeral secrets (LES) Compromise of ephemeral keys of target session should not compromise session key.

Advanced properties for key agreement

Forward secrecy Compromise of long term keys of both parties *after session is complete* should not compromise session key.

Key compromise impersonation (KCI) resistance The adversary should not be able to masquerade to A as any different party B.

Leakage of ephemeral secrets (LES) Compromise of ephemeral keys of target session should not compromise session key.

Advanced properties for key agreement

Forward secrecy Compromise of long term keys of both parties *after session is complete* should not compromise session key.

Key compromise impersonation (KCI) resistance The adversary should not be able to masquerade to A as any different party B.

Leakage of ephemeral secrets (LES) Compromise of ephemeral keys of target session should not compromise session key.

Weak forward secrecy

- A weaker form of forward secrecy is provided when the adversary is not allowed to be active in the target session.
- There is a common myth that full forward secrecy is not possible for a one-round (two message) key agreement protocol.
- It is true that one-round protocols with full forward secrecy do not exist in strong models.

Outline

Notation

p	A large prime (usually at least 1024 bits).
q	A prime (usually of at least 160 bits) with $q \mid p-1$.
G	A subgroup of $\mathbb{Z}_p^*$ often of order q.
g	A generator of G.
r_A, r_B	Random integers chosen by A and B respectively.
t_A, t_B	Ephemeral public keys: typically $t_A = g^{r_A}$ and $t_B = g^{r_B}$.
x_A, x_B	Private long-term keys of A and B respectively.
y_A, y_B	Public keys of A and B: $y_A = g^{x_A}$ and $y_B = g^{x_B}$.
Z_{AB}	Shared secret calculated by the principals.
S_{AB}	Static Diffie-Hellman Key of A and B: $S_{AB} = g^{x_A x_B}$.

MTI protocols

- A number of protocols were suggested by Matsumoto, Takashima and Imai in 1986, which incorporate authentication into the Diffie-Hellman exchange in an elegant manner.
- Three classes of MTI protocols were proposed. They are all of the same basic format; they involve only two messages and achieve implicit key authentication but no key confirmation.
- Basic protocols are vulnerable to some instructive attacks.

Three types of MTI protocol

Type	z_{AB}	z_{BA}	Z_{AB}	Computed by A	Computed by B
A(0)	g^{r_A}	g^{r_B}	$g^{x_A r_B + x_B r_A}$	$z_{BA}^{x_A} y_B^{r_A}$	$z_{AB}^{x_B} y_A^{r_B}$
B(0)	$y_B^{r_A}$	$y_A^{r_B}$	$g^{r_A + r_B}$	$z_{BA}^{x_A^{-1}} g^{r_A}$	$z_{AB}^{x_B^{-1}} g^{r_B}$
C(0)	$y_B^{r_A}$	$y_A^{r_B}$	$g^{r_A r_B}$	$z_{BA}^{x_A^{-1} r_A}$	$z_{AB}^{x_B^{-1} r_B}$

- In the table z_{AB} and z_{BA} are the messages sent from A to B, and from B to A, respectively.
- For each basic protocol there are an infinite number of derived protocols in that type.

Small subgroup attacks

- These attacks are possible if the base point g is of composite order and allow an active adversary to force the shared secret into a small subgroup generated by a power of g.
- This attack applies to the MTI protocol C(0) in the situation that the group G is the whole of $\mathbb{Z}_p^*$ as originally proposed.
- We suppose that the factorisation of $p-1$, which is the order of G, is known to the adversary.

Small subgroup attack on MTI C(0)

- The attack is easiest in the case that $p-1$ has a very small factor r; let us write $w = (p-1)/r$.
- The attack works by raising the exchanged messages to the power w which moves these elements into the small subgroup of G of order r.
- The adversary C plays in the middle between A and B.

$$
\begin{array}{lll}
1. & A \to C_B: & y_B^{r_A} \\
1'. & C_A \to B: & y_B^{r_A w} \\
2. & B \to C_A: & y_A^{r_B} \\
2'. & C_B \to A: & y_A^{r_B w}
\end{array}
$$

- The shared secret calculated by A and B is

$$Z_{AB} = g^{r_B r_A w}.$$

- Since this is an element in the small subgroup, C can easily find the shared secret by exhaustive search. Notice that in the extreme case $r = 1$ and then $w = p - 1$ so the element received by both A and B is 1.
- If g is chosen to have order q which is prime, these attacks do not apply, as long as it is ensured that exchanged elements are not the identity in the group.

Lim and Lee attack on MTI Protocols

- Lim and Lee (1997) devised attacks on interactive protocols that work in prime order subgroups. Their attack is applicable to MTI variants in which G is a prime order subgroup.
- The idea of the attack is that the adversary will send in message 1 a value that is not in G, and consequently the key calculated by B will give away information about B's long-term secret key x_B.
- In an interesting echo of the prime order subgroup attack, this requires that $(p-1)/q$ consist of a product of small factors.

Lim and Lee attack on MTI protocol A(0)

1. $A \rightarrow B: \quad \beta t_A$
2. $B \rightarrow A: \quad t_B$

- Suppose that β is an element whose order is a small factor r of $p-1$.
- The shared secret is calculated by B as $Z_{BA} = (\beta t_A)^{x_B} y_A^{r_B}$. Since $t_A^{x_B} = y_B^{r_A}$ and $y_A^{r_B} = t_B^{x_A}$, A can calculate Z_{BA}/β^{x_B} and there are only r possible values for β^{x_B}. A can try out each of these in turn.
- There are a number of ways that A is able to know if the correct value has been found.
- To complete the attack A repeats this procedure with new factors of $(p-1)/q$ in place of r until the value of x_B is obtained.

Avoiding the attack of Lim and Lee

1. Each recipient of a protocol message can check that the message lies in G. The cost of this is an exponentiation which is a significant extra computational burden.
2. Choose the prime p so that $(p-1)/q$ has no small factors apart from 2. In this case the attack will give away one bit of information about the principals' secrets.

Unknown key-share attacks

- These attacks work if the adversary can make one party complete the protocol and believe the key is shared with B while it is shared with a different party.
- Menezes, Qu and Vanstone discovered unknown key-share attacks on all the classes of MTI protocols.
- The attacks require the adversary C to obtain a certificate for a long-term key y_C which is related to the public key of A by the equation

$$y_C = y_A^{x_C} = g^{x_A x_C}.$$

- Note that C cannot know the corresponding private key $x_A x_C$ of this public key.

Unknown key-share attack on MTI protocol B(0)

$$
\begin{array}{lll}
1. & A \to C_B: & y_B^{r_A} \\
1'. & C \to B: & y_B^{r_A} \\
2. & B \to C: & y_C^{r_B} \\
2'. & C_B \to A: & (y_C^{r_B})^{x_C^{-1}} = y_A^{r_B}
\end{array}
$$

- The shared secret calculated by A is $(y_A^{r_B})^{x_A^{-1}} g^{r_A} = g^{r_B + r_A}$ while B calculates $(y_B^{r_A})^{x_B^{-1}} g^{r_B} = g^{r_A + r_B}$ to get the same value.
- Although A and B both have the same session key, A believes it to be shared with B, while B believes it to be shared with C.

Avoiding unknown key-share attacks

- Certifiers of public keys could ensure that each entity is in possession of the corresponding private key before a certificate is issued.
- Key confirmation can defeat the attack. The confirmation messages should include the identities of both principals.
- Include both principal identities within the key derivation function. If the function used is collision resistant, when A believes the key is shared with C she will always derive a different key from B who believes the key is shared with A.

Key compromise impersonation

- Suppose the long term key of A is compromised by an adversary. A key compromise impersonation (KCI) attack is one in which the adversary is able to masquerade *to* A as another party.
- Consider the protocol MTI C(0). If the adversary C knows x_A then he can use the first message sent by A to construct an attack.

$$\begin{array}{lll} 1. & A \to C_B: & y_B^{r_A} \\ 2. & C_B \to A: & y_B^{x_A r_C} \end{array}$$

- A then calculates the shared secret as $y_B^{r_A r_C}$ which can also be calculated by C. Therefore protocol C(0) is vulnerable to key compromise impersonation.

Outline

Signed Diffie–Hellman

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
	$\xleftarrow{t_B, Sig_B(t_B, t_A, A)}$	
Verify signature		$Z_{AB} = t_A^{r_B}$
$Z_{AB} = t_B^{r_A}$	$\xrightarrow{Sig_A(t_A, t_B, B)}$	Verify signature

The Unified model

- This protocol, and the MQV protocol below, are part of the IEEE P1363 standard and NIST SP 800-56A (2007).
- The name of the protocol comes from the unification of the static and ephermeral Diffie-Hellman keys.
- The protocol is attributed to Ankney, Johnson and Matyas.
- The shared secret is the concatenation of the static and ephemeral Diffie-Hellman keys: $Z_{AB} = g^{r_A r_B} \parallel g^{x_A x_B}$.

Unified model protocol

A		**B**
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
	$\xleftarrow{t_B}$	
$Z_{AB} = t_B^{r_A} \parallel S_{AB}$		$Z_{AB} = t_A^{r_B} \parallel S_{AB}$

- The session key for UM in NIST standard is defined as $K = KDF(Z_{AB} \parallel A \parallel B \parallel \ldots)$ for a suitable key derivation function KDF.

Before accepting the shared key A must make the following checks. B makes the analogous checks.

1. $1 < t_B < p$. In particular, degenerate values such as 0 and p should not be allowed.
2. $t_B^q \bmod p = 1$. This ensures that $Z_{AB} \in G$ as long as A has chosen t_A correctly.
3. $Z_{AB} \neq 1$. Together with the previous check this ensures that Z_{AB} has order q so preventing any small order subgroup attacks.

(H)MQV protocol

- Protocol due originally to Menezes, Qu and Vanstone (1995) with later improvements including Law and Solinas with these authors.
- MQV makes use of a special operation on elements of $\mathbb{Z}_p$ which limits the length to a fixed size w. Typically w would be 80. The result of the operation on $t \in \mathbb{Z}_p$ is denoted $\bar{t} = t \bmod 2^w + 2^w$.
- The shared secret is: $Z_{AB} = g^{S_A S_B} \bmod p$ where $S_A = r_A + \bar{t}_A x_A \bmod q$ and similarly $S_B = r_B + \bar{t}_B x_B \bmod q$.
- HMQV due to Krawczyk (2005) is a variant protocol which defines $\bar{t}_A = H(t_A, B)$ and $\bar{t}_B = H(t_B, S)$ where H is a random oracle.

(H)MQV protocol

A		**B**
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
	$\xleftarrow{t_B}$	
$S_A = r_A + \bar{t}_A x_A \bmod q$		$S_B = r_B + \bar{t}_B x_B \bmod q$
$Z_{AB} = (t_B y_B^{\bar{t}_B})^{S_A}$		$Z_{AB} = (t_A y_A^{\bar{t}_A})^{S_B}$

- The same checks by each principal as in the Unified Model are required before accepting the shared key.
- The session key for HMQV is defined as $K = KDF(Z_{AB})$ for a suitable key derivation function KDF.

NAXOS protocol

A		**B**
$r_A \leftarrow \{0,1\}^k$		
$h_A = H_1(x_A, r_A)$		
$t_A = g^{h_A}$	$\xrightarrow{t_A}$	$r_B \leftarrow \{0,1\}^k$
		$h_B = H_1(x_B, r_B)$
		$t_B = g^{h_B}$
	$\xleftarrow{t_B}$	
$Z_{AB} = H_2(t_B^{x_A}, y_B^{h_A}, t_B^{h_A}, A, B)$		$Z_{AB} = H_2(y_A^{h_B}, t_A^{x_B}, t_A^{h_B}, A, B)$

- Note that Z_{AB} is the session key.
- NAXOS allows ephemeral secrets to be revealed even in the target session.

Comparison

Properties → ↓ *Protocol*	Fwd. secrecy	Resists KCI	LES	Exponentiations off-line	Exponentiations on-line
Signed DH	Yes	Yes	No	1	1 + sig
UM	Weak	No	Yes	2	1
HMQV	Weak	Yes	Yes	1	1.5
NAXOS	Weak	Yes	Yes	2	2

- Off-line exponentiations can be performed before the message exchange begins, assuming public keys are available.
- For signed Diffie–Hellman “sig” means both signature generation and verification
- Some properties require validity checking

Proofs for Key Establishment

Colin Boyd

Department of Telematics
NTNU

May 2014

NTNU – Trondheim
Norwegian University of
Science and Technology

The need for formality

- Until the 1990s it was common for the security goals of key establishment (KE) protocols to be defined informally (e.g. entity authentication, implicit key authentication, freshness, forward secrecy, resistance to dictionary attacks)
- Hence the security analysis of these protocols has also been informal.
- Many published protocols have been found flawed years after their publication.
- As with other cryptographic primitives, we would like to know precisely the security properties that KE protocols achieve and the underlying computational assumptions.

Computational security of KE protocols

Initiated by Bellare and Rogaway (1993). Followed up by:

- server-based protocols: Bellare–Rogaway (1995)
- public-key based key transport: Blake-Wilson–Menezes (1997)
- key agreement protocols: Blake-Wilson–Menezes (1998)
- password-based protocols: Bellare–Pointcheval–Rogaway (2000)
- group key agreement: Bresson–Chevassut–Pointcheval (2001)
- RFID protocols: Vaudenay (2007)
- browser-based protocols: Gajek, Manulis, Sadeghi, Schwenk (2008)

At the same time many model variants and enhancements were proposed.

Computational models

The methodology follows the common computational reductionist approach.

- Adversary model - What can the adversary do?
- Notions of security - When does the adversary break the security of the protocol?
- Proofs by reduction – if the adversary breaks the security of the protocol, then the adversary can break some (assumed) intractable computational problem.

Symbolic models

There is a large alternative body of work outside the cryptographic community using idealized notions of cryptography.

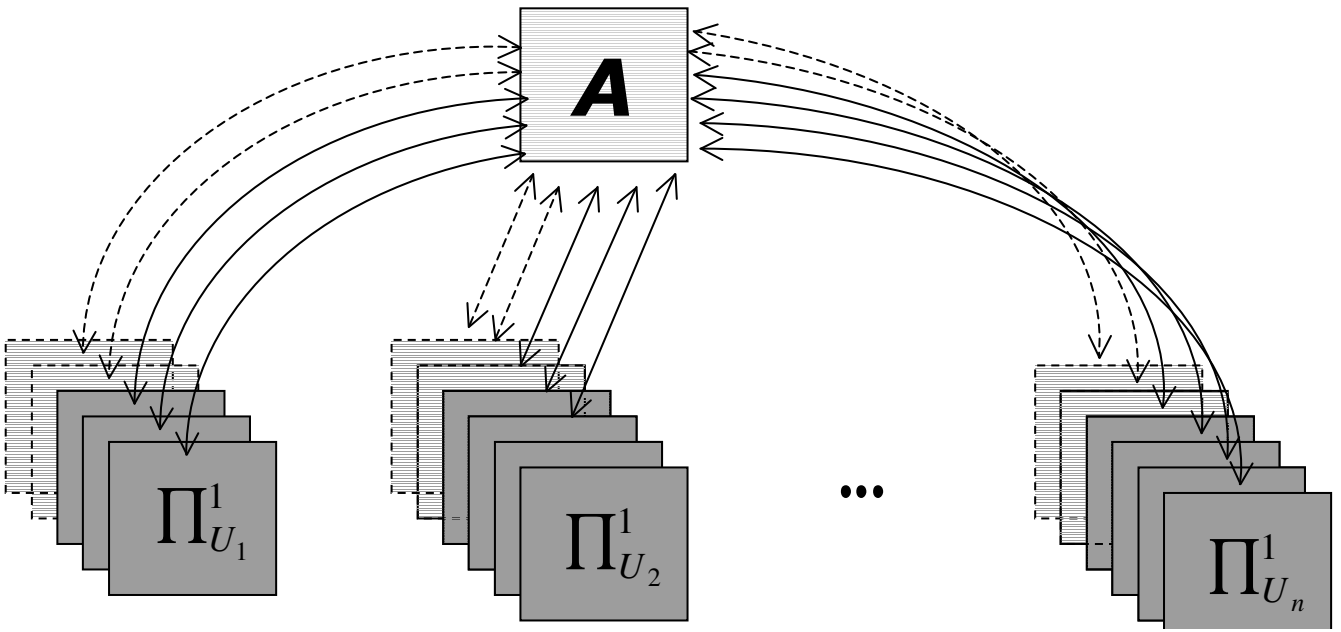
- Often credited back to 1983 paper of Dolev and Yao
- Often use tool support for automatic analysis of protocols
- Applicable to both *model checking* and *theorem proving* approaches.
- Modern tools in this mould include ProVerif and Scyther
- Recent trends include unification of computational models and machine support (e.g. CryptoVerif).

In these slides we only look at computational models.

Bellare and Rogaway's security model

- The adversary controls all the communications that take place and does this by interacting with a set of *oracles*, each of which represents an instance of a principal in a specific protocol run.
- The principals are defined by an identifier U from a finite set and an oracle Π^s_U represents the actions of principal U in the protocol run indexed by integer s.
- Principals' long-term keys are initialised using a key generation algorithm LL.
- Interactions with the adversary are called oracle *queries*.
- Security of protocols is defined in terms of indistinguishability of established session keys from random keys.

- The adversary is computationally bounded to probabilistic polynomial time.



Oracle queries

Send(U, s, M)	Send message M to oracle Π_U^s
Reveal(U, s)	Reveal session key (if any) accepted by Π_U^s
Corrupt(U, K)	Reveal state of U and set long-term key of U to K
Test(U, s)	Attempt to distinguish session key accepted by oracle Π_U^s

This list applies to the Bellare and Rogaway's 1995 paper; additional queries are appropriate in other models.

Send(U, s, M)

- Allows the adversary to make the principals run the protocol normally. The oracle Π^s_U will return to the adversary the next message that an honest principal U would do if sent message M according to the conversation so far.
- If Π^s_U accepts the session key or halts this is included in the response. The adversary can also use this query to start a new protocol instance by sending a special message '*start: U′*' in which case U will start a protocol run with a new index s and partner ID U'.

Reveal(U, s)

This query models the adversary's ability to find old session keys. If a session key K_s has previously been accepted by Π_U^s then it is returned to the adversary. An oracle can only accept a key once (of course a principal can accept many keys modelled in different oracles).

Corrupt(U, K)

This query models insider attacks by the adversary. The query returns the oracle's internal state and sets the long-term key of U to be the value K chosen by the adversary. The adversary can then control the behaviour of U with Send queries.

Reveal(U, s)

This query models the adversary's ability to find old session keys. If a session key K_s has previously been accepted by Π_U^s then it is returned to the adversary. An oracle can only accept a key once (of course a principal can accept many keys modelled in different oracles).

Corrupt(U, K)

This query models insider attacks by the adversary. The query returns the oracle's internal state and sets the long-term key of U to be the value K chosen by the adversary. The adversary can then control the behaviour of U with Send queries.

Test(U, s)

Once the oracle Π_U^s has accepted a session key K_s the adversary can attempt to distinguish it from a random key as the basis of determining security of the protocol. A random bit b is chosen; if $b = 0$ the K_s is returned while if $b = 1$ a random string is returned from the same distribution as session keys.

In the 1993 and 1995 papers of Bellare–Rogaway it was stated that Test(U, s) must be the final query of the adversary. This condition is not a good model for security and was later removed.

Test(U, s)

Once the oracle Π_U^s has accepted a session key K_s the adversary can attempt to distinguish it from a random key as the basis of determining security of the protocol. A random bit b is chosen; if $b = 0$ the K_s is returned while if $b = 1$ a random string is returned from the same distribution as session keys.

In the 1993 and 1995 papers of Bellare–Rogaway it was stated that Test(U, s) must be the final query of the adversary. This condition is not a good model for security and was later removed.

Adversary's advantage

- Security of a KE protocols is defined based on a game played by the adversary $\mathcal{A}$.
- Success of the adversary is measured in terms of its *advantage* in distinguishing the session key from a random key after running the Test query. If we define Good-Guess to be the event that the adversary guesses correctly whether $b = 0$ or $b = 1$ then

$$\mathrm{Adv}_{\mathcal{A}} = \left| \Pr[\text{Good-Guess}] - \frac{1}{2} \right| .$$

Freshness

The Test query may only be used for a fresh oracle. An oracle is said to be *fresh* when:

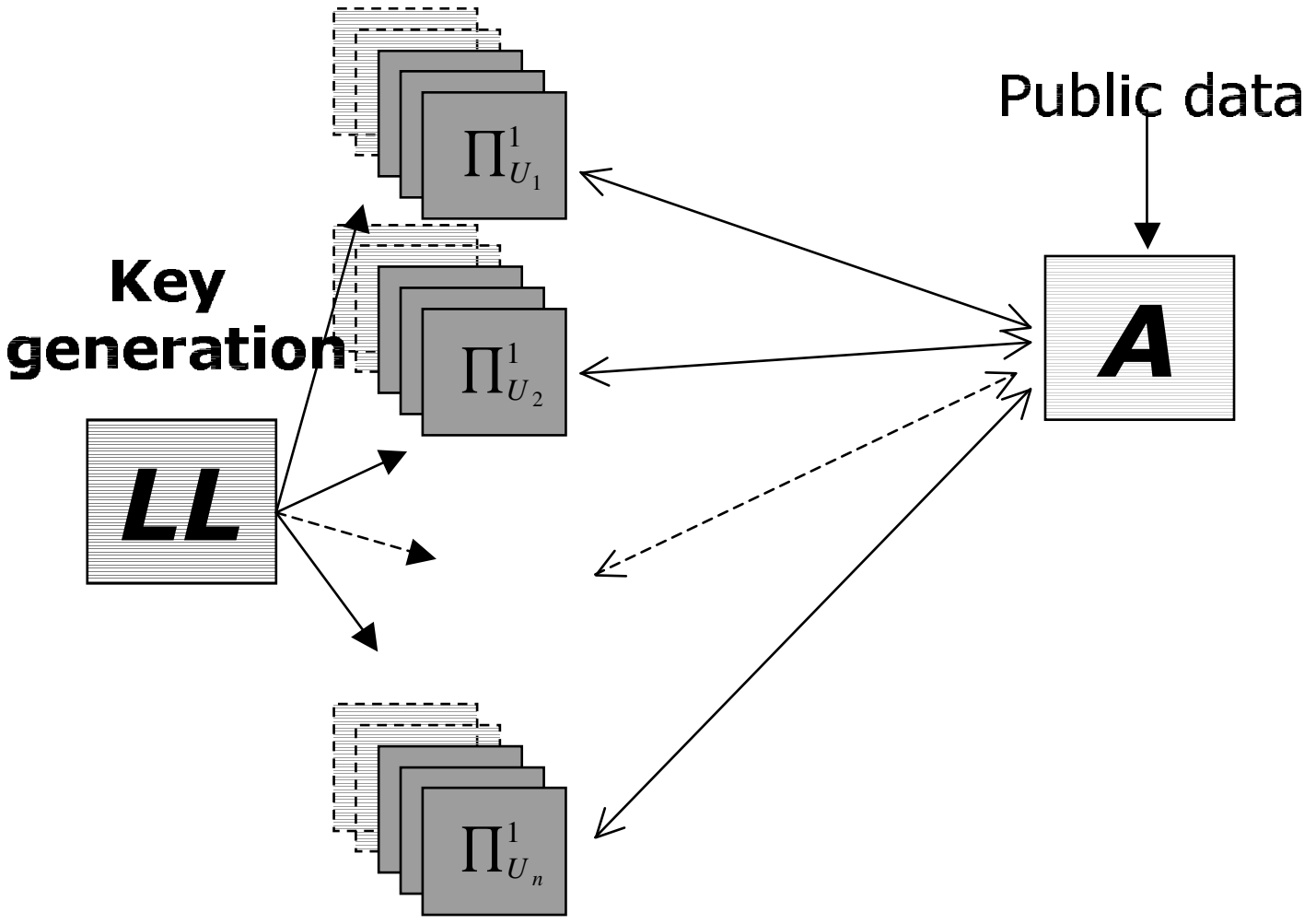
- it has accepted a session key, and
- neither itself nor the *partner* oracle have had a Corrupt or Reveal query.
- The way of defining partner oracles has varied in different papers. We will assume existence of a *session identifier (SID)* which is a unique value known to each partner.
- Partners must both have accepted the same session key, have the same SID, and recognise each other as partners.

The security game

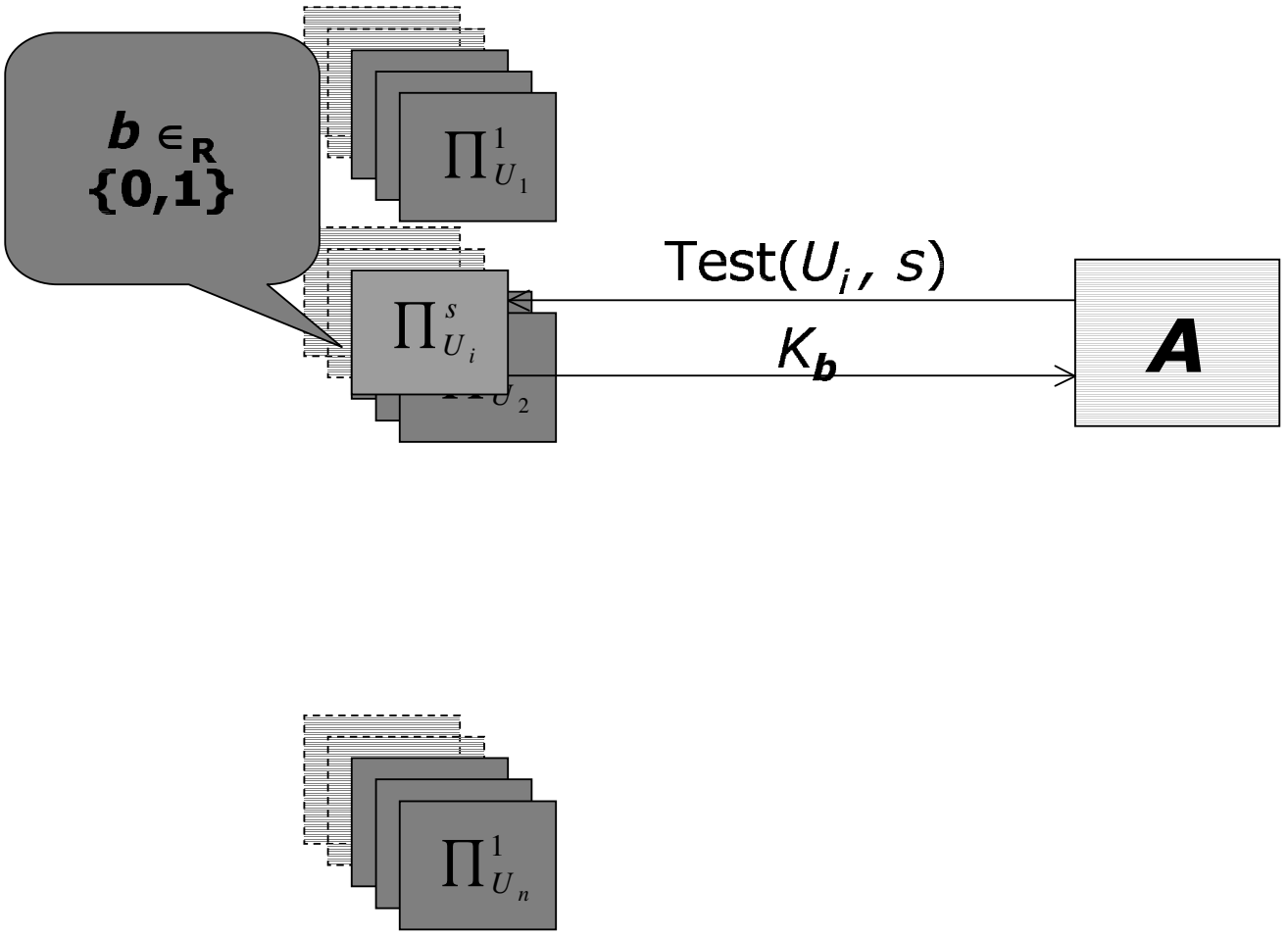
The game played by the adversary $\mathcal{A}$ consists of three stages:

- **Stage 1:** All principals are initialised with long-term keys using the key generation algorithm $\mathcal{LL}$. Any public data (e.g. public keys) are handed to $\mathcal{A}$, who the interacts with the principals via queries.
- **Stage 2:** $\mathcal{A}$ chooses a fresh oracle $\Pi^s_{U_i}$ and queries it with Test(U_i, s). A key K_b is returned to the adversary.
- **Stage 3:** $\mathcal{A}$ is allowed to continue asking queries to the oracles, but is not allowed to reveal the test session or corrupt the principals involved in the test session. At the end of this stage $\mathcal{A}$ outputs its guess b'.

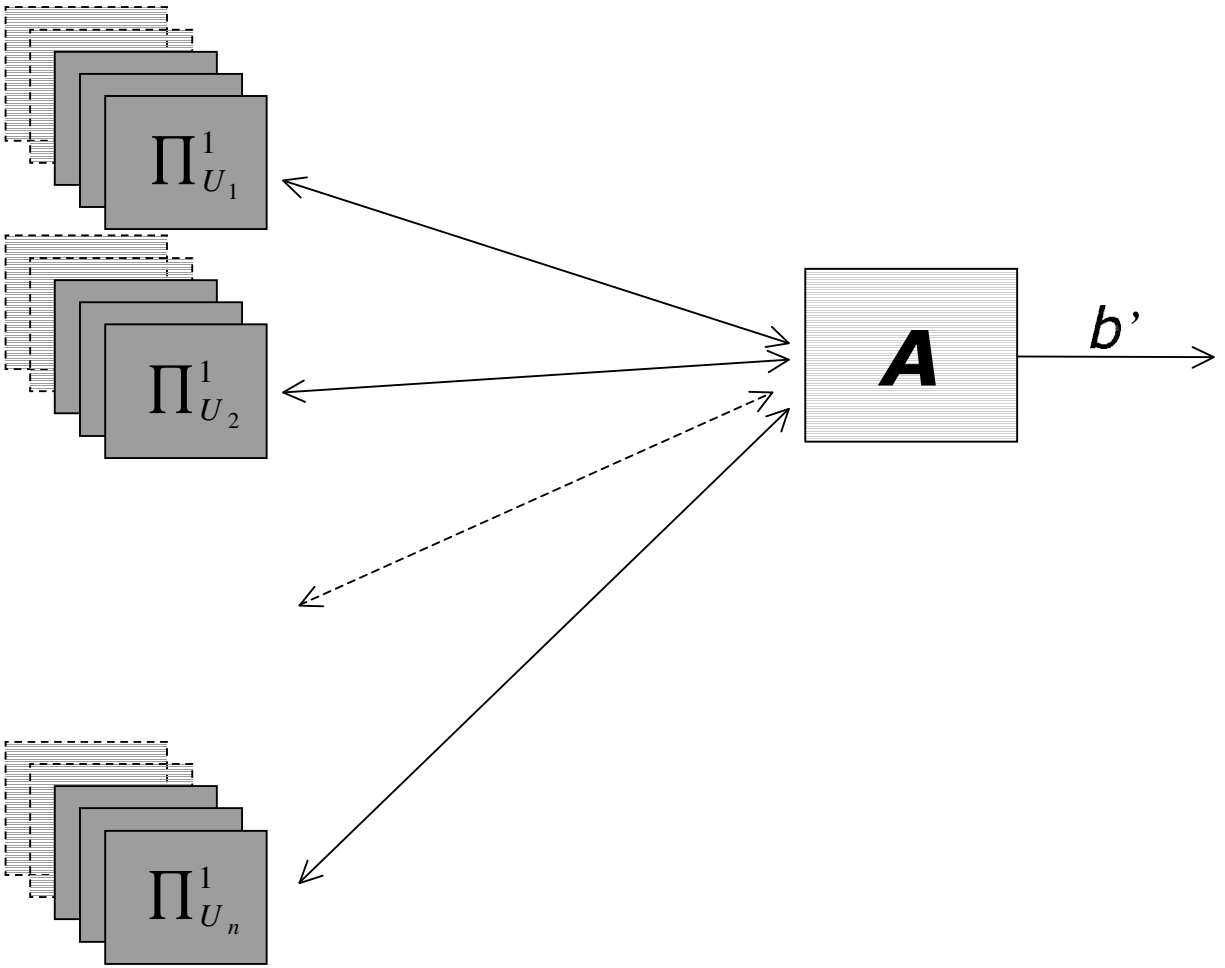
Stage 1



Stage 2



Stage 3



Definition of security

- We say that an adversary is *benign* if it simply relays messages between oracles.
- Bellare and Rogaway define a protocol in this model to be secure if:
  1. when the protocol is run by a benign adversary both principals will accept the same session key.
  2. $\mathrm{Adv}_{\mathcal{A}}$ is negligible for all (probabilistic polynomial time) adversaries.

- The first condition is a completeness criterion that guarantees that the protocol will complete as expected in normal circumstances.
- The second condition says that the adversary is unable to find anything useful about the session key.
- Although this definition appears to be concerned only with key confidentiality it does imply key authentication.
  - Suppose that the session key is known to an oracle Π_U^s different from that recorded in an oracle $\Pi_{U'}^t$ to be tested.
  - Π_U^s is not the partner of $\Pi_{U'}^t$ and so it can be opened by the adversary and so the protocol cannot be secure.

Example

- In their 1995 paper Bellare and Rogaway proved the security of a server based protocol, similar to that examined earlier.
- A trusted server generates a random session key K_{AB} and sends it securely to two principals protected with long-term keys.
- The protocol uses a symmetric-key encryption algorithm and a MAC algorithm.
- We show that as long as the encryption mechanism and MAC are secure then the adversary cannot obtain a significant advantage, hence the protocol is also secure.

3PKD Protocol

1. $A \to B: \quad A, B, N_A$
2. $B \to S: \quad A, B, N_A, N_B$
3. $S \to A: \quad N_B, \mathcal{E}_{K_{AS}}(K_{AB}), \mathcal{M}_{K'_{AS}}(A, B, N_A, N_B, \mathcal{E}_{K_{AS}}(K_{AB}))$
4. $S \to B: \quad \mathcal{E}_{K_{BS}}(K_{AB}), \mathcal{M}_{K'_{BS}}(A, B, N_A, N_B, \mathcal{E}_{K_{BS}}(K_{AB}))$

- Keys K_{AS} and K'_{AS} are independent.
- $(\mathcal{E}(\cdot), \mathcal{D}(\cdot))$ is an IND-CPA symmetric-key encryption scheme.
- $\mathcal{M}(\cdot)$ is a MAC unforgeable under adaptive chosen-message attacks (UF-CMA).
- We use the value $N_A \parallel N_B$ as the session identifier.

IND-CPA symmetric encryption

Recall the game played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. $\mathcal{C}$ chooses a random key K.

1. **Find stage:** $\mathcal{A}$ can ask for the encryption of any message m of its choice. $\mathcal{C}$ replies with $\mathcal{E}_K(m)$
2. $\mathcal{A}$ chooses *any* two equal length messages m_0 and m_1. $\mathcal{C}$ chooses random bit $b \in_R \{0, 1\}$ and gives $c = \mathcal{E}_K(m_b)$ to $\mathcal{A}$.
3. **Guess stage:** $\mathcal{A}$ can continue to request for the encryption of any message m of its choice. $\mathcal{C}$ replies with $\mathcal{E}_K(m)$. Finally $\mathcal{A}$ outputs a bit b'.

The adversary's advantage is:

$$Adv_{\mathcal{A}} = \left| \Pr(b' = b) - \frac{1}{2} \right|$$

Double encryption

- In the protocol 3PKD the adversary always sees the encryption of the same message under two different keys K and K'
- Hence, we need to consider an extended definition of IND-CPA, where the adversary has access to an encryption oracle that when queried on a message m, returns two ciphertexts $(\mathcal{E}_K(m), \mathcal{E}_{K'}(m))$.
- The double encryption adversary $\mathcal{DE}$ outputs two messages m_0 and m_1 at the end of the 'find' stage and gets $c = (\mathcal{E}_K(m_b), \mathcal{E}_{K'}(m_b))$, where $b \in_R \{0, 1\}$.
- The adversary $\mathcal{DE}$ breaks the IND-CPA security of the double encryption scheme if it guesses b with non-negligible advantage.

One can prove the following result needed in the proof of security of the key establishment protocol:

Theorem

Let $\mathcal{E}$ be a IND-CPA secure encryption scheme, then the double encryption scheme built from $\mathcal{E}$ is IND-CPA secure.

The proof is left as an exercise.

UF-CMA MAC

Consider the following game played between a challenger $\mathcal{C}$ and an adversary $\mathcal{F}$:

1. $\mathcal{C}$ chooses a random key K.
2. $\mathcal{A}$ can ask for the tag (MAC value) of any message m of its choice. $\mathcal{C}$ replies with $\mathcal{M}_K(m)$. Similarly, $\mathcal{F}$ can check the validity of any pair of message and MAC tag (m, τ) of its choice. $\mathcal{C}$ replies with $\mathcal{V}_K(m, \tau)$ (either 'true' or 'false')
3. $\mathcal{F}$ wins if it can output a valid tag on a message that has not been previously output by the challenger.

The MAC scheme is unforgeable under adaptive chosen message attacks (UF-CMA) if Pr($\mathcal{F}$ wins) is negligible for all PPT adversaries.

Proof security of 3PKD (sketch)

Let $\mathcal{A}$ be an adversary against the 3PKD protocol that succeeds with probability $\text{Succ}_{\mathcal{A}}$ when the protocol is run with n_p principals. Let n_s be the maximum number of sessions between any two principals. Both n_p and n_s are polynomial functions.

We consider two cases:

1. $\mathcal{A}$ gains her advantage by forging a MAC with respect to some user's MAC key;
2. $\mathcal{A}$ gains her advantage without forging a MAC.

Case 1: Adaptive MAC forger $\mathcal{F}$

- Assume that at some point $\mathcal{A}$ outputs a forgery of a MAC with respect to some user's MAC key that $\mathcal{A}$ does not know.
- More precisely, we define the event forge at some point in the game:
  - $\mathcal{A}$ asks a Send query which includes a message with a valid MAC;
  - $\mathcal{A}$ has not corrupted the owner of the MAC key;
  - the message and MAC have not been output as the answer to a Send query.
- We construct an adaptive forger algorithm $\mathcal{F}$ against the MAC scheme that uses $\mathcal{A}$.
- $\mathcal{F}$ has access to the MAC generation and verification oracles associated with the MAC key x' .

$\mathcal{F}$ runs $\mathcal{A}$ on a simulated interaction with a set of n_p principals, as follows:

- chooses one of the principals, I, at random;
- generates all encryption and MAC keys randomly except for the MAC key corresponding to Π_I;
- responds to all of $\mathcal{A}'s$ queries as per 3PKD specification;
- to simulate messages involving Π_I, $\mathcal{F}$ uses the MAC generation oracle;
- if a Corrupt query is sent to Π_I then the algorithm fails. This only happens if we are 'unlucky'.

Success probability of $\mathcal{F}$

- $\mathcal{F}$ monitors $\mathcal{A}$'s queries to detect any forgery (using the verify oracle).
- Note that, except when we are 'unlucky', the simulation of the principals by $\mathcal{F}$ is perfect: $\mathcal{A}$ cannot "discriminate" against any particular principal; they all look the same.
- When event forge occurs, the probability that it corresponds to Π_{ij} is $1/n_p$. In this case $\mathcal{F}$ wins its MAC game by outputting the same MAC. This happens with probability

$$\Pr(\mathrm{Succ}_{\mathcal{F}}) = \Pr(\mathrm{forge})/\mathrm{n_p}.$$

Case 2: Double-Encryption attacker $\mathcal{DE}$

- Assume now that event forge does not occur.
- Then we construct an algorithm $\mathcal{DE}$ that breaks the IND-CPA security of the double encryption scheme.
- $\mathcal{DE}$ has access to the double encryption oracle corresponding to keys x and y.
- $\mathcal{DE}$ chooses a random pair of messages m_0 and m_1 of length equal to that of session keys in 3PKD, and hands them to the challenger.
- $\mathcal{DE}$ receives $c = (\mathcal{E}_x(m_b), \mathcal{E}_y(m_b))$, where $b \in_R \{0, 1\}$

$\mathcal{DE}$ runs $\mathcal{A}$ on a simulated interaction with a set of n principals, as follows:

- chooses randomly two oracles Π_I^s and Π_J^t which are instances of principals I and J.
- $\mathcal{DE}$ generates all encryption and MAC keys randomly except for those for the encryption keys of principals I and J.
- $\mathcal{DE}$ can answer directly all queries to principals except for I and J because $\mathcal{DE}$ generated their keys.
- For encryption queries to principals I and J, $\mathcal{DE}$ uses the encryption oracles available from the IND-CPA game, except for the sessions Π_I^s and Π_J^t.

Simulating queries for $\mathcal{DE}$

- Send queries to the server for Π_i^s and Π_j^t are answered with $\mathcal{E}_x(m_b), \mathcal{E}_y(m_b)$, respectively, as the encrypted session keys.
- If a Reveal or Corrupt query is sent to Π_i^s and Π_j^t then the algorithm $\mathcal{DE}$ aborts and returns a random bit. This only happens if we are 'unlucky'.
- Because there is no MAC forgery, every ciphertext output by $\mathcal{A}$ was generated properly. This means it was actually generated by $\mathcal{DE}$ so Reveal queries can be answered correctly for oracles other than Π_i^s and Π_j^t.
- If the Test query points to Π_i^s or Π_j^t then $\mathcal{DE}$ gives $\mathcal{A}$ plaintext m_0 and return the same answer that $\mathcal{A}$ does. This happens when we are 'lucky' and has polynomial (non-negligible) probability.

Advantage of $\mathcal{DE}$

If algorithm $\mathcal{DE}$ chooses the right test session, which happens with probability $\frac{1}{n_p^2 n_s}$, then $\mathcal{DE}$ succeeds whenever $\mathcal{A}$ succeeds. Hence:

$$\begin{aligned}\Pr(\mathrm{Succ}_{\mathcal{DE}}) &= \frac{1}{n_p^2 n_s} \cdot \Pr(\mathrm{Succ}_{\mathcal{A}} | \overline{\mathrm{forge}}) + \left(1 - \frac{1}{n_p^2 n_s}\right) \cdot 1/2 \\ &= 1/2 + \frac{1}{n_p^2 n_s} \left(\Pr(\mathrm{Succ}_{\mathcal{A}} | \overline{\mathrm{forge}}) - \frac{1}{2}\right)\end{aligned}$$

We can rewrite this as:

$$\Pr(\mathrm{Succ}_{\mathcal{A}} | \overline{\mathrm{forge}}) - \frac{1}{2} = n_p^2 n_s \cdot Adv_{\mathcal{DE}}$$

Conclusion of proof

The proof concludes by observing that:

$$\begin{aligned} Adv_{\mathcal{A}} &= \Pr(\mathrm{Succ}_{\mathcal{A}}) - 1/2 \\ &\leq \Pr(\mathrm{forge}) + \Pr(\mathrm{Succ}_{\mathcal{A}} | \overline{\mathrm{forge}}) - 1/2 \\ &\leq n_p \Pr(\mathrm{Succ}_{\mathcal{F}}) + n_p^2 n_s \mathrm{Adv}_{\mathcal{DE}} \end{aligned}$$

It follows that if the advantage of $\mathcal{A}$ is non-negligible, then we can either forge MACs with non-negligible probability or break the encryption scheme with non-negligible advantage.

Models for Key Establishment

Colin Boyd

Department of Telematics
NTNU

May 2014

NTNU – Trondheim
Norwegian University of
Science and Technology

Outline

Bellare–Rogaway model evolution

- BR93 covered mutual authentication and key exchange from pre-shared symmetric-keys.
- BR95 covered server-based key exchange (Kerberos style)
- Blake-Wilson and Menezes (1997, 1998) extended model to cover public keys and key agreement.
- BPR00 (B, R and Pointcheval) covered password-based protocols and also extended model to include forward secrecy.

CK model evolution

- BCK98 introduces modular approach and secure channel goals
- CK01 fixes up problems with BCK98
- CK-HMQV 2005 model fixes up problems with CK01 but ignores modular approach
- eCK introduced simple model, capturing some things CK-HMQV cannot, leaving other things out.
- CF12 puts back forward secrecy into eCK.

Differentiators for computational models

- How partnering is defined / what freshness means
- What the adversary is allowed to obtain
- When the adversary is allowed to obtain things

Models we will not go into

- Shoup (1997)
  - Simulation based (possibly too strong)
  - Not much used
- Password authenticated key exchange (PAKE)
- Group key exchange
- Timestamps
- Universal composability (UC)

Outline

Canetti-Krawczyk (CK01) model

- Similar basic idea to Bellare–Rogaway models
- Builds on Bellare, Canetti and Krawczyk (1998)
- Two main motivations:
  - build secure channels for sessions
  - a modular design approach using *authenticators*.
- Allows *session state* to be revealed

Sessions

- Partnering in the CK01 model is defined through session identifiers.
- Any party P_i starts a protocol run when it receives an input of the form $(P_i, P_j, s, \text{role})$ for a session identifier s and $\text{role} \in \{\text{initiator}, \text{responder}\}$.
- Two sessions $(P_i, P_j, s, \text{role})$ and $(P_j, P_i, s', \text{role}')$ are said to be *matching* if $s = s'$.
- It is not required that $\text{role}' \neq \text{role}$

Adversary queries

The following special queries are available to the adversary.

Party corruption. This query returns the long-term key of the party and also all the memory which may include ephemeral keys or session keys.

Session key reveal. As in the BR models the adversary can obtain the session key of any completed session by asking a reveal query.

Session state reveal. The session-state reveal query can be asked of an incomplete session and returns the internal state. The model allows the protocol to specify what is included in the session state, but a typical example would be an ephemeral Diffie–Hellman exponent.

Session expire. This query deletes the session key from the input session specified.

Freshness in CK01

- A session $(P_i, P_j, s, \text{role})$ is *fresh* as long as:
  - it has not been asked a session-state reveal query;
  - it has not been asked a reveal query;
  - P_i was not asked a corrupt query before the session was asked an expire query;
  - the above three conditions also hold for any matching session $(P_j, P_i, s, \text{role}')$.
- The condition on corrupt captures forward secrecy.
- The adversary is only allowed to choose fresh sessions as its test session.

Security game

1. Long-term keys are generated for all parties using a protocol-dependent function called initialization which is run before the protocol starts.
2. The adversary can send messages to any instance and observe the response.
3. The adversary can ask: reveal queries to any instance; session-state reveal queries to any incomplete instance; or expire queries to any complete instance.
4. The adversary can ask corrupt queries to any principal.
5. At some point the adversary asks a test query to any fresh instance. The adversary can continue to send message and ask other queries as long as the tested session remains fresh.
6. Eventually the adversary outputs its guess bit b'.

CK01 security definition

Definition

A protocol Π is a *secure authenticated key exchange protocol* in the CK01 model if both of the following hold:

1. if two uncorrupted parties complete matching sessions then both instances will end in the accept state with the same session key;
2. the advantage of any efficient adversary $\mathcal{A}$ in guessing the correct bit in the security game is negligible.

Modular design

- Define two worlds:
  - *Authenticated links model* or AM: the adversary $\mathcal{A}$ is not able to fabricate messages but can only activate instances using messages output by legitimate parties.
  - *Unauthenticated links model* or UM: a full adversary is allowed.
- A protocol π which is secure in the AM can be transformed into a protocol $\mathcal{C}(\pi)$ which is secure in the UM be applying a valid authenticator $\mathcal{C}$.
- Secure protocols in the AM can generally be much simpler than protocols in the UM.
- Secure protocols in the AM can be defined independently of authenticators but then combined to obtain secure protocols in the UM.

Post-specified peers

- Variant of CK01 published in 2002 by Canetti and Krawzyk
- Some protocols do not require both parties to identify at the start of the protocol. This can help with anonymity.
- CK02 is a relaxed model with a new definition of partnering which allows session to have uncompleted partner sessions.
- They prove the IKE protocol, used in IPSec, secure in this model
- Later, Menezes and Ustaoglu show that the HMQV protocol is not secure in the post-specified setting.

Criticisms of CK model

The CK01 model has received some significant criticism.

Session identifiers. The usage of session identifiers has been one of the most controversial aspects of the CK01 model because there is no concrete definition. Instead, the CK01 paper states that session identifiers are: . . . chosen by a “higher layer” protocol that “calls” the protocol.

Session state query What constitutes session state can be defined by the protocol designer. Not necessary realistic.

Restrictions on queries No session which has had a session-state reveal query can be fresh. This rules out some attacks which are captured in other models since the test session cannot have its ephemeral keys revealed.

Criticisms of CK model

The CK01 model has received some significant criticism.

Session identifiers. The usage of session identifiers has been one of the most controversial aspects of the CK01 model because there is no concrete definition. Instead, the CK01 paper states that session identifiers are: . . . chosen by a “higher layer” protocol that “calls” the protocol.

Session state query What constitutes session state can be defined by the protocol designer. Not necessary realistic.

Restrictions on queries No session which has had a session-state reveal query can be fresh. This rules out some attacks which are captured in other models since the test session cannot have its ephemeral keys revealed.

Criticisms of CK model

The CK01 model has received some significant criticism.

Session identifiers. The usage of session identifiers has been one of the most controversial aspects of the CK01 model because there is no concrete definition. Instead, the CK01 paper states that session identifiers are: . . . chosen by a “higher layer” protocol that “calls” the protocol.

Session state query What constitutes session state can be defined by the protocol designer. Not necessary realistic.

Restrictions on queries No session which has had a session-state reveal query can be fresh. This rules out some attacks which are captured in other models since the test session cannot have its ephemeral keys revealed.

HMQV model

Based on CK01 with a few enhancements.

- A session at principal ID_A with intended partner B has session identifier $(\mathrm{ID}_A, \mathrm{ID}_B, \mathit{Out}, \mathit{In})$ where In and Out are the messages received and sent by the session.
- Matching is then defined the same as in the original CK so that completed sessions are matched if and only if their identifiers are of the form $(\mathrm{ID}_A, \mathrm{ID}_B, \mathit{Out}, \mathit{In})$ and $(\mathrm{ID}_B, \mathrm{ID}_A, \mathit{In}, \mathit{Out})$.
- Key compromise impersonation (KCI) attacks are captured by allowing the adversary to obtain the private key of the owner of the test session.
- The session-state reveal query comes in two flavours.
  1. The session state is defined as empty except for the session key which can be obtained with a normal reveal query.
  2. Later the session-state reveal includes the ephemeral secret key chosen for that session.

Outline

The eCK model

- Proposed in 2007 by LaMacchia, Lauter and Mityagin. They called their model an Extended Canetti–Krawczyk model and it is now widely referred to as the eCK model.
- Tackles directly some of the perceived weaknesses in the CK and BR models. Specific advantages are:
  - the adversary can obtain ephemeral secrets which belong to the test session;
  - the adversary can obtain the long-term key of the test session and of its partner even before the session is completed.
- Was widely believed that the eCK model is strictly stronger than the CK01 (or HMQV) model, but there are other features which the eCK model does not capture.

A generic one-round protocol

private key x_A		private key x_B
A		**B**
$r_A \leftarrow R$		
$t_A = f(x_A, r_A, \text{public info})$		
	$\xrightarrow{t_A}$	
		$r_B \leftarrow R$
		$t_B = \hat{f}(x_B, r_B, \text{public info})$
	$\xleftarrow{t_B}$	
$Z_{AB} = F(x_A, r_A, t_B, \text{public info})$		$Z_{AB} = \hat{F}(x_B, r_B, t_A, \text{public info})$

- r_A and r_B are *ephemeral secrets*
- Z_{AB} is the *shared secret*

Idea of the eCK model

The general idea behind the eCK model is simple and appealing.

- Each party in a protocol run has two secrets - a long-term secret x_A and an ephemeral secret r_A, the latter chosen for this particular protocol run.
- The adversary should be allowed to obtain any of these secrets which do not trivially allow the session key to be computed.
- So the adversary should be allowed to obtain (r_A, r_B) or (x_A, x_B) or (r_A, x_B) or (x_A, r_B) but not (r_A, x_A) or (r_B, x_B).

Freshness in eCK model

- As in the CK01 model, partnering is defined through session identifiers. However, sessions are only partners if they agree on which one of them takes the initiator and which takes the responder role.
- A session with ephemeral key r_A at party A with intended partner B is *fresh* as long as:
  - the session has not been asked a reveal query;
  - if a matching session exists at B with ephemeral key r_B then:
    - both r_A and x_B were not revealed;
    - both r_B and x_B were not revealed;
  - if no partner exists then x_B has not been revealed.

Naxos protocol (again)

$y_A = g^{x_A}$	$y_B = g^{x_B}$
A	**B**
$r_A \leftarrow R$	
$h_A = H_1(x_A, r_A)$	
$t_A = g^{h_A}$	
$\xrightarrow{t_A}$	
	$r_B \leftarrow R$
	$h_B = H_1(x_B, r_B)$
	$t_B = g^{h_B}$
$\xleftarrow{t_B}$	
$Z_{AB} = H_2(t_B^{x_A}, y_B^{h_A}, t_B^{h_A}, A, B)$	$Z_{AB} = H_2(y_A^{h_B}, t_A^{x_B}, t_A^{h_B}, A, B)$

The NAXOS trick

- NAXOS protocol is proven secure in eCK model.
- Notice that basic ephemeral Diffie–Hellman, authenticated in any way, is insecure in the eCK model.
- In the NAXOS protocol, the ephemeral secret key is combined with the long-term key, preventing the adversary from obtaining anything useful from the ephemeral reveal query.
- This idea is now widely known as the *NAXOS trick* and there is some disagreement about whether using the trick is in some sense cheating.

Comparing models

- It seems to be accepted now that CK and eCK models are incomparable. (Doubt is around what is the precise definition of state reveal queries in CK.)
- Cremers (2011) shows that differences in partnering is enough to separate all three models.
- Cremers (2009) gives a neat attack on Naxos when session state includes all the parameters except the long-term key. Ostaoglu (2009) showed that a similar attack is available on HMQV.

Separating eCK from HMQV (Cremers 2011)

- In eCK, sessions must agree on their roles in order to be partners.
- In HMQV, sessions are partners if they have the same session ID, independent of whether they agree on their roles.
- A protocol like NAXOS includes the IDs of the participants in the KDF. This means that NAXOS cannot be secure in CK model.
- A protocol like HMQV does not include the IDs of the participants in the KDF. This means that HMQV cannot be secure in eCK model.
- Arguably this is a detail which just needs to be taken care of: sessions key definition and matching definitions must line up.

Outline

Defining strong and weak forward secrecy

A protocol provides forward secrecy if the adversary cannot distinguish the session key from a random string even given the long-term keys of both parties after the session is complete.

Strong forward secrecy (sFS)

This property holds even if the adversary takes an active part in the session under attack. In this case the victim executes the session with the adversary who masquerades as a legitimate party.

Weak forward secrecy (wFS)

This property only holds when the adversary is prevented from taking an active part in the session under attack. In this case the victim executes the session with a legitimate party whose messages are transmitted correctly to the victim.

Diffie–Hellman-based one-round protocol (DH1R)

private key x_A
A
$r_A \leftarrow R$
$t_A = g^{r_A}$

$\xrightarrow{t_A}$

private key x_B
B
$r_B \leftarrow R$
$t_B = g^{r_B}$

$\xleftarrow{t_B}$

$Z_{AB} = F(x_A, r_A, t_B, \text{public info})$

$Z_{AB} = \hat{F}(x_B, r_B, t_A, \text{public info})$

- Here g generates a group in which Diffie–Hellman is secure
- Prominent concrete protocols include MQV, HMQV and UM

DH-based protocol cannot have strong FS

Krawczyk attack on any DH1R protocol

- Active adversary chooses valid r_A and sends t_A masquerading as A.
- B accepts and may encrypt secrets for A.
- After expiry adversary obtains x_A and can compute shared secret in same way as A would have.

Krawczyk showed instead that HMQV provides weak forward secrecy

Gone viral

Krawczyk, Crypto 2005

"We note that this form of attack can be carried against *any* 2-message key-exchange protocol (including HMQV) authenticated via public keys and with no secure shared state previously established between the parties."

Menezes and Ustaoglu, ASIACCS, 2008

"Bellare, Pointcheval and Rogaway (2000) (see also [HMQV]) observed that two-pass key agreement protocols can only achieve weak forward secrecy . . . "

Saar et al., SCN 2010

"no two-pass key exchange protocol can achieve forward secrecy"

Extending Krawczyk's attack

1. $\mathcal{A}$ observes and records the first message t_A from A to B.
2. $\mathcal{A}$ asks for the ephemeral value r_A used in this instance.
3. $\mathcal{A}$ initiates a new instance of the protocol with B. In this instance $\mathcal{A}$ masquerades as A.
4. $\mathcal{A}$ sends t_A from the earlier protocol instance as the first message of the new instance.
5. B will compute a new r_B and t_B and send t_B to A which is captured by $\mathcal{A}$.
6. $\mathcal{A}$ eventually expires the session and corrupts A to obtain x_A.
7. $\mathcal{A}$ now has all the inputs of F so can re-compute Z_{AB}.

Impact of attack

- Attack applies to the generic one round protocol.
- The adversary can recover any information sent by B during the session established in step 3.

Proposition (BG 2011)

A one-round key exchange protocol cannot provide strong forward secrecy in any model which allows the adversary to reveal ephemeral secrets of the partner party to the test session. This holds even if ephemeral secrets cannot be revealed during the test session.

Cremers–Feltz models

- eCK model cannot capture full forward secrecy because it has no notion of timing.
  Cremers and Feltz (2012, 2013) defined an *origin session* of the test session to be one which *sends* the message received by the test session.
- They define a model called eCK-PFS which extends eCK by allowing the adversary to obtain either:
  - the long-term keys after the session is complete when the adversary is active and can choose the peer ephemeral key; or
  - exactly one of (the ephemeral key of an origin session) and (the long-term key of the peer) when an origin session exists.
- Cremers and Feltz (2012, 2013) apply signatures, while BG (2012) apply MACs, to achieve strong forward secrecy generically for one-round protocols.

Some key exchange challenges

- Can we unify existing models in any useful way? Or perhaps classify them?
- Is there a stronger or a strongest model? Is there a “right” model?
- Can different protocol goals be usefully included in key exchange models? Examples might be key renegotiation or ciphersuite and version negotiation. Why was 3-shake not captured in TLS analysis?
- Still a lot of real-world protocols whose security is not formally analysed.

Outline

Standard model

- Widely believed a good thing to avoid the random oracle model (ROM).
- Most efficient schemes like HMQV rely on ROM for their proof.
- KDF can be made standard model by using PRF, for example with Krawczyk's HMAC-based construction.
- Generic constructions exist which can use generic standard model primitives.

Incorporating certification systems

- ASICS model of BCFPPS at ESORICS 2013
- Give the adversary access to public key registration query.
- Includes the registration of *invalid* keys. This captures some well known real-world attacks.
- Allow users to own multiple public keys and also the same public key for multiple identities.
- A framework for incorporating certification into common game-based AKE models.

Leakage resilient key exchange

- Leakage resilience is a new trend in cryptography since around 2005. Aims to capture side channel attacks.
-
  - $\mathcal{A}$ gets access to a chosen function of the long-term secret with some restrictions
  - Leakage can be continuous or bounded
  - Leakage can be restricted to before the target session occurs
  - Best model (if such exists) not yet established.
- Natural to include key exchange in the list of leakage-resilient primitives.
- First results by Moriyama and Okamoto, 2011.

Recent results (ASB 2014)

- Strengthen BR or eCK model by giving the adversary access to leakage function.
- Generic constructions can be made using existing leakage-resistant primitives.
- Plug LR-encryption into generic encryption-based key exchange construction.
- To achieve *continuous* leakage resilience need to have stateful protocols

Secure channels

In the key exchange models security has been defined to be essentially as strong as we can imagine. However, we can ask for more.

1. The key exchange protocol and application in practice run in parallel so we should worry about analyzing the security of the combination of key exchange and applications.
2. In some applications, prominently TLS, we are unable to achieve the strong indistinguishability definition for key exchange.

ACCE is a currently popular example of a weakened BR-style security definition for secure channels suitable for proving security of TLS.

Post quantum

- Post quantum cryptography is current important trend.
- Lattice-based encryption algorithms based on LWE implicitly include a Diffie-Hellman type of key agreement protocol which is post-quantum secure.