Proofs for Key Establishment Protocols

Colin Boyd

Information Security Institute
Queensland University of Technology

December 2007

Outline

Purpose of key establishment

- Two or more networked parties wish to establish secure communications
- In order to use cryptography they must share a key, usually called a *session key*
- Necessary to have some existing infrastructure in place:
  - an existing (long-term) shared key
  - a public key infrastructure (certificate-based or ID-based)
  - a mutually trusted third party

Types of key establishment protocol

- Two parties or group of parties
- Key agreement or key transport
- Server-based or server-less

Goals of key establishment

- Key authentication: each party knows who has the session key
- Entity authentication: each party knows which other parties are active
- Known-key security: the adversary may have obtained session keys from other 'old' sessions
- Forward secrecy: the adversary may learn the long-term secrets of the parties after the session has run
- Key compromise impersonation (KCI) security: the adversary may learn the long-term key of one of the parties before the session has run

Analysis of key establishment protocols

- Traditionally the security goals of key establishment protocols have been defined and analysed informally.
- Many published protocols have been found flawed, even years after their publication.
- As with other cryptographic primitives, we would like to know precisely the security properties that protocols achieve and the underlying comptutational assumptions.

Outline

Proofs for key establishment

Initiated by Bellare and Rogaway (1993). Follows the typical computational approach used for provable (reductionist) proofs:

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

General approach

A mathematical model defines a protocol in which a powerful adversary plays a central role.

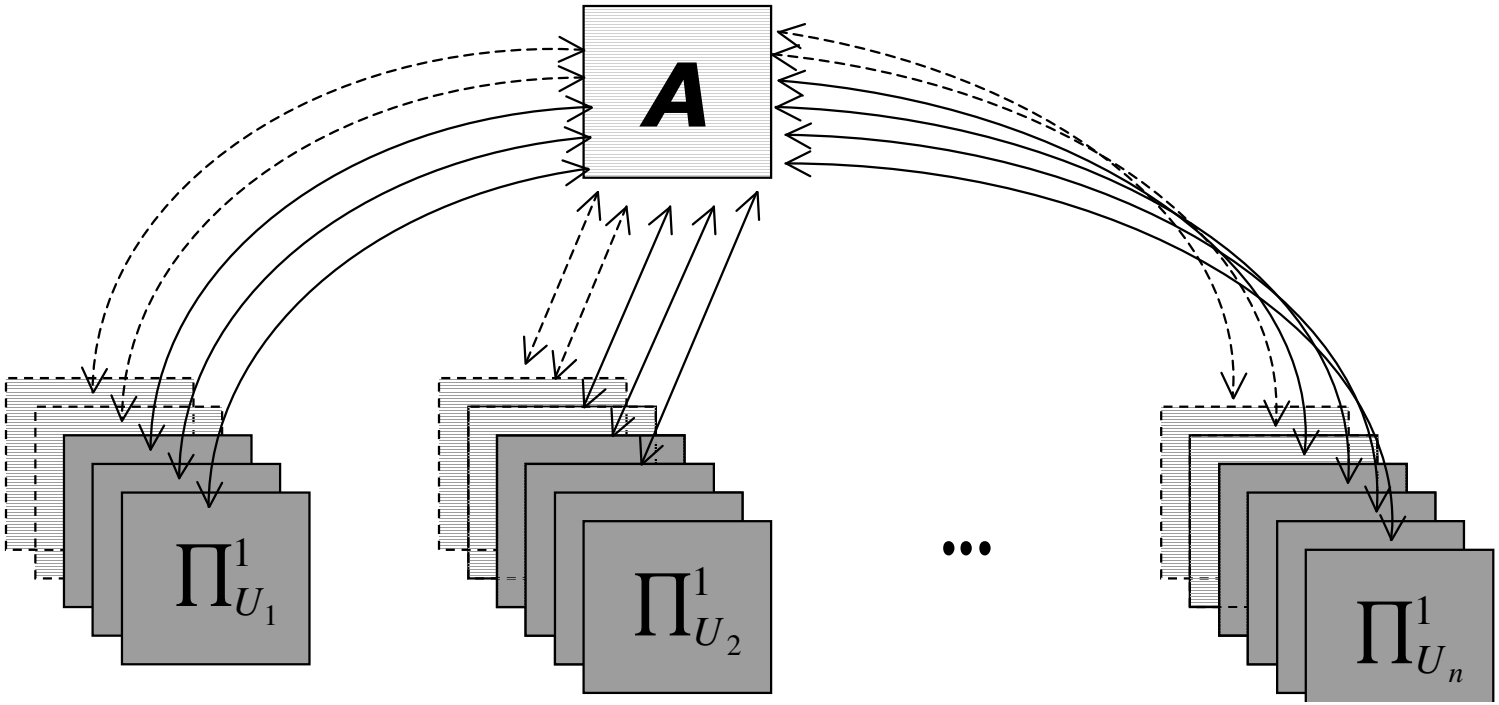
- Adversary controls all the principals and can initiate protocol runs between any principals at any time.
- Adversary can alter and fabricate messages using anything that it can compute
- Insider attacks are modelled by allowing the adversary to corrupt any principals, and the adversary can also obtain previously used keys.
- Security of protocols is defined in terms of indistinguishability of established session keys from random keys.

Security model

The model is the same for all protocols with the same set of principals and the same protocol goals.

- The adversary controls all the communications 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_U^s 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*.

- The adversary is computationally bounded to probabilistic polynomial time.



Outline

Oracle queries

- Send(U,s,M): Send message M to oracle Π_U^s
- Reveal(U,s): Obtain the session key (if any) accepted by Π_U^s
- Corrupt(U,K): Obtain the current 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

Send(U, s, M)

- Allows the adversary to make the principals run the protocol normally. The oracle Π_U^s 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_U^s 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 an empty message M in which case U will start a protocol run with a new index s.

Reveal(U, s)

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)

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.

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.

Partnering

- The way of defining partner oracles has varied in different papers. More recently partners have been defined by having the same *session identifier (SID)* which consists of a concatenation of the messages exchanged between the two.
- Two sessions (oracles) are partners if both have accepted, have the same SID and recognise each other as partners.

Outline

Adversary's advantage

- Security of a protocol is defined based on a game played by the adversary $\mathcal{A}$.
- Success of $\mathcal{A}$ 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 the advantage of $\mathcal{A}$ is

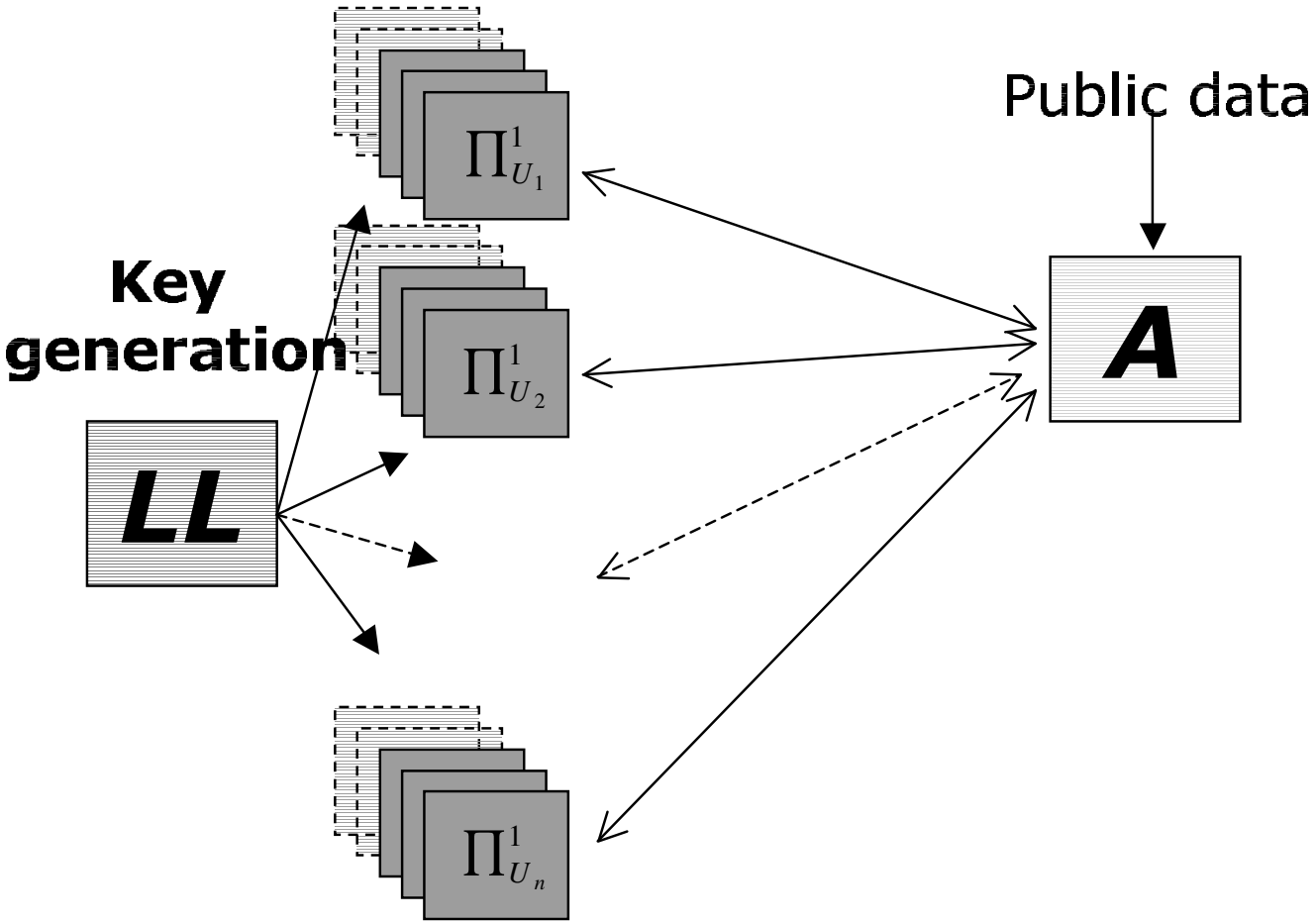
$$\mathrm{Adv}^{\mathcal{A}} = 2 \times |\Pr[\text{Good-Guess}] - \frac{1}{2}|.$$

The adversary game

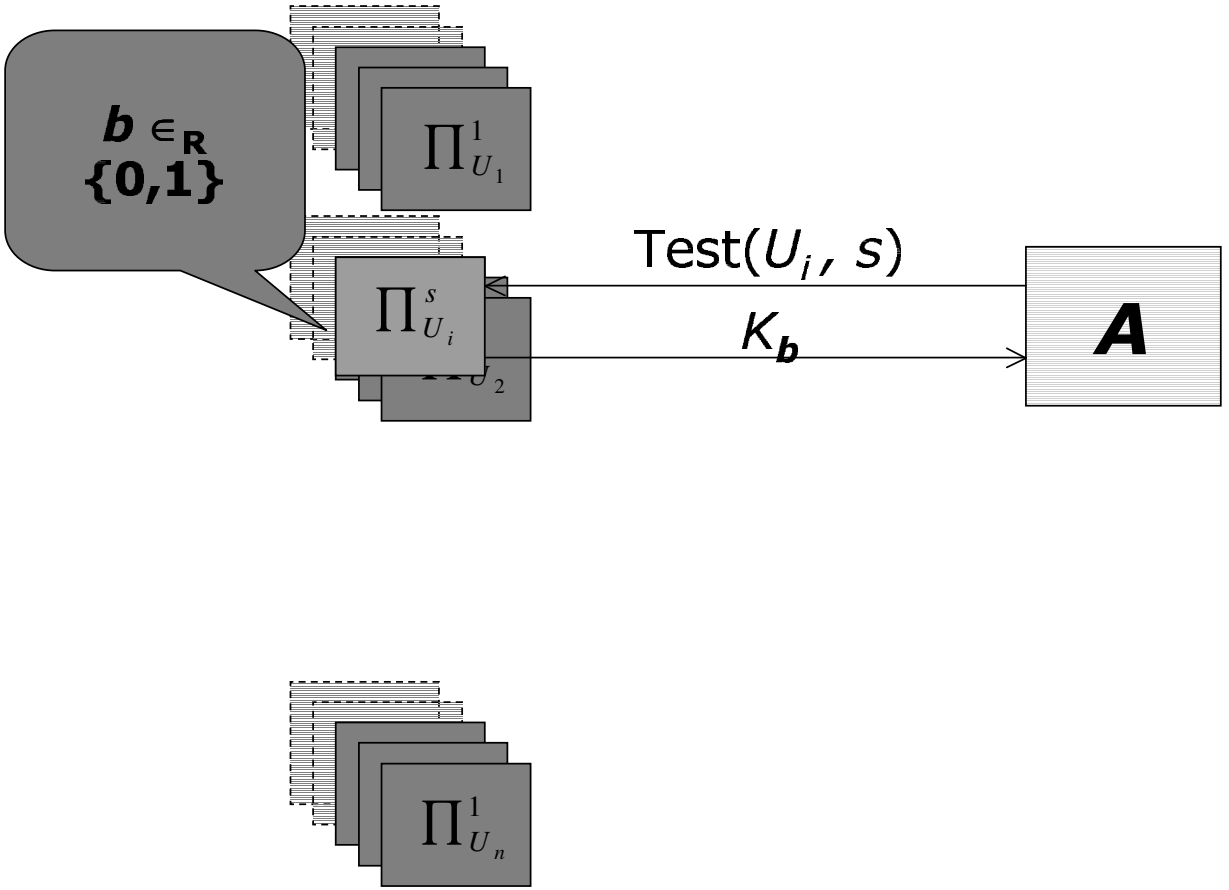
The game played by the adversary $\mathcal{A}$ consists of three stages:

- **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 then 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'.

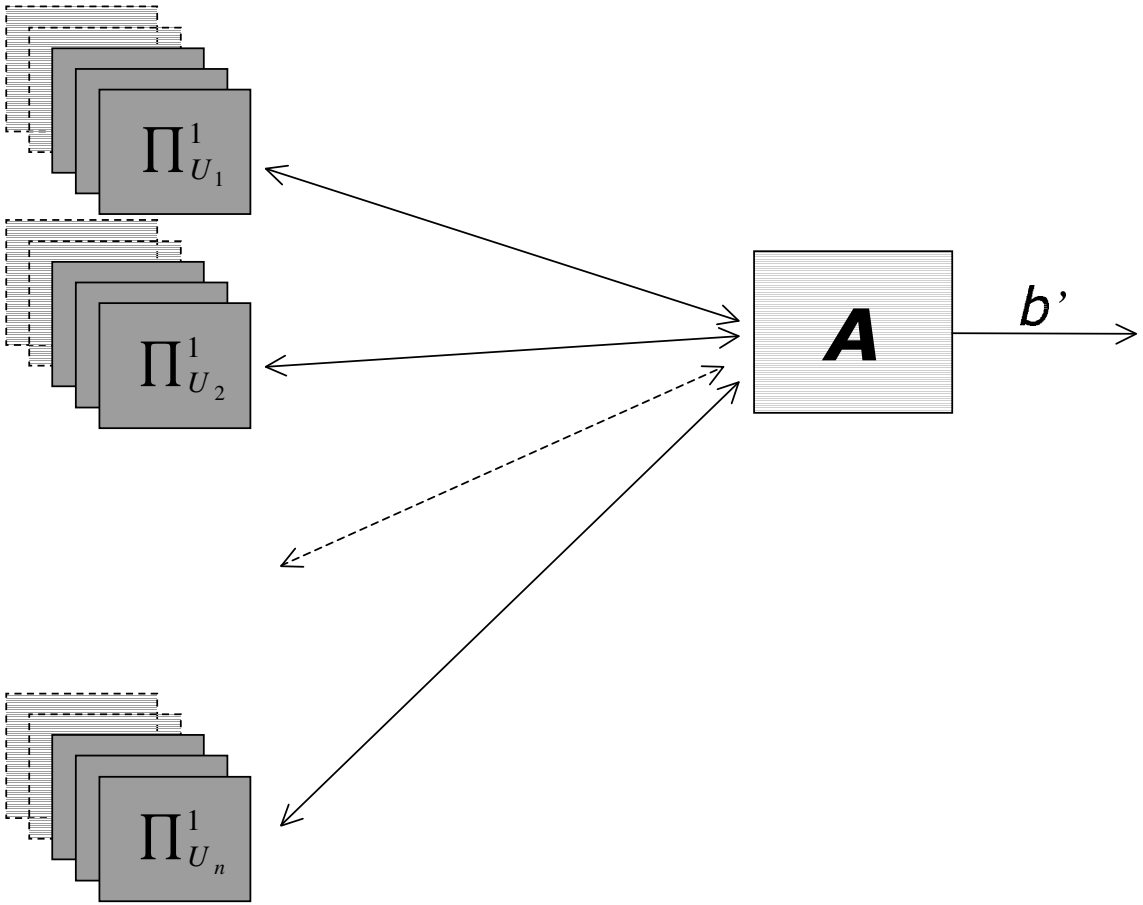
Stage 1



Stage 2



Stage 3



Definition of security

We say that an adversary is *benign* if it simply relays messages between oracles.

Definition

A protocol is a *secure key establishment protocol* if:

1. when the protocol is run by a benign adversary both principals will accept the same session key.
2. $\text{Adv}^{\mathcal{A}}$ 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 (3PKD) similar to Kerberos.
The protocol uses the following cryptographic primitives:

- a CPA-secure symmetric-key encryption algorithm $\mathcal{E}_K(\cdot)$.
- a secure MAC algorithm $\mathcal{M}_K(\cdot)$. More precisely, we require $\mathcal{M}$ to be secure against *adaptive chosen-message attacks*.

Outline

Protocol setting

- The protocol involves two users (principals), A and B, and a server, S. It enables any two users to obtain a new session key.
- The server initially shares a two secret key with all principals. The keys shared between A and S are denoted K_{AS} and K'_{AS} and are assumed independent.
- The server chooses a new session key and distributes it to the two users A and B.

3PKD protocol

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

Theorem

Assume that the encryption algorithm is CPA-secure and the MAC scheme is secure against adaptive chosen message attacks. Then the 3PKD protocol is a secure key establishment protocol.

Proof strategy

The proof proceeds by contraction.

1. Assume that the adversary $\mathcal{A}$ can obtain a significant advantage in the game
2. Build a new adversary to break one of the cryptographic primitives. The new adversary uses $\mathcal{A}$ as a subroutine.
3. Since we assume that the cryptographic primitives are secure it follows that $\mathcal{A}$ cannot exist.

In order to achieve this goal a *simulator* of the oracles must be defined which is efficiently constructed from public information.

Outline

Preliminaries: CPA secure encryption

An encryption scheme $\mathcal{E}_K(.)$ is secure under chosen plaintext attack (CPA secure) if there is no efficient adversary that can win the following game.

Find stage

- The adversary is given access to an oracle that will return encryptions of messages of its choice.
- The adversary chooses two messages m_0 and m_1.

Guess stage

- The adversary is given a ciphertext $C = \mathcal{E}_K(m_b)$ where b is a random bit.
- The adversary returns a bit b' and wins the game if $b' = b$.

Preliminaries: multiple encryption

- In the protocol the adversary sees the encryption of the same message (K_{AB}) under two different keys (K_{AS} and K_{BS}).
- Hence, we need to consider an extended definition of CPA security, where the adversary has access to an encryption oracle $\mathcal{O}_{K_{AS},K_{BS}}(\cdot)$ that when queried on a message m, returns two ciphertexts

$$(\mathcal{E}_{K_{AS}}(m), \mathcal{E}_{K_{AS}}(m)) = \mathcal{O}_{K_{AS},K_{BS}}(m).$$

- The multiple-encryption adversary outputs two messages m_0 and m_1 at the end of the 'find' stage. The input to the 'guess' stage is $(c_0, c_1) = \mathcal{O}_{K_{AS},K_{BS}}(m_b)$, where $b \in_R \{0, 1\}$. The adversary breaks the multiple encryption scheme if it guesses b with non-negligible advantage.

One can prove the following result needed in the proof of security of the 3PKD protocol.

Lemma

If $\mathcal{E}$ is a CPA-secure encryption scheme then any efficient multiple-encryption adversary has negligible advantage.

Preliminaries: MACs

- A *message authentication code* (MAC) is a tag that is appended to a message to provide data integrity and authentication.
- A MAC is a function that takes as input a message M and a key K and outputs the MAC value $\mathcal{M}_K(M)$.
- The verification algorithm takes as input a message, a key and a claimed MAC and has boolean output.

MAC security

Security of MACs is defined by the following game.

- An adversary $\mathcal{A}$ is given access to a MAC oracle that provides valid MAC on messages of $\mathcal{A}$'s choice.
- Access to the oracle is *adaptive* in the sense that $\mathcal{A}$ can choose its inputs after seeing previous answers.
- $\mathcal{A}$ outputs a claimed forgery and wins the game if the forgery is valid.

A MAC is secure against adaptive chosen message attacks if there is no efficient adversary that can win the game with non-negligible advantage.

Outline

Sketch of proof for 3PKD

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 $\mathcal{A}$ outputs a valid MAC in some Send query that it was not previously given as the result of some other query.
- We construct an adaptive forger algorithm $\mathcal{F}$ against the MAC scheme that uses $\mathcal{A}$.
- $\mathcal{F}$ is provided permanent access to the MAC oracle $\mathcal{OM}_{x'}$ associated with the MAC key x'
- The goal of $\mathcal{F}$ is to output a valid MAC that it was not given.

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

- chooses one of the principals, Π_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 $\mathcal{OM}_{X'}$;
- if a Corrupt query is sent to Π_I then the algorithm fails. This only happens if we are 'unlucky'.

- When $\mathcal{A}$ produces a forged MAC, $\mathcal{F}$ returns the message-MAC pair produced by $\mathcal{A}$.
- 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.
- Let ν_f be the probability that $\mathcal{A}$ forges a MAC. When this occurs, the probability that it correspond to Π_I is $1/n_p$.
- Therefore $\mathcal{F}$ will obtain a forgery on a message by Π_I with probability

$$\text{Succ}^{\mathcal{F}} = \nu_f / n_p.$$

Case 2: Multiple-Encryption attacker $\mathcal{ME}$

- Assume $\mathcal{A}$ gains an advantage against 3PKD without forging a MAC.
- Then we construct an algorithm $\mathcal{ME}$ that breaks the IND-CPA security of the multiple-encryption scheme.
- $\mathcal{ME}$ is provided permanent access to the multiple-encryption oracle $\mathcal{O}_{x,y}(\cdot)$.
- $\mathcal{ME}$ 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.
- $\mathcal{ME}$ receives $(c_0, c_1) = \mathcal{O}_{x,y}(m_b)$, where $b \in_R \{0, 1\}$

$\mathcal{ME}$ 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.
- $\mathcal{ME}$ generates all encryption and MAC keys randomly except for those for the encryption keys of principals I and J.
- $\mathcal{ME}$ can answer all queries to principals except for I and J because $\mathcal{ME}$ generated their keys.

- Send queries to the server for Π_I^s and Π_J^t are answered with c_0 and c_1,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 fails. This only happens if we are 'unlucky'.
- If the Test query points to Π_I^s or Π_J^t then give $\mathcal{A}$ the plaintext m_0 and return the same answer that $\mathcal{A}$ does. This happens when we are 'lucky' and has polynomial (non-negligible) probability.
- The MAC ensures that we are not 'unlucky' with overwhelming probability. If A could forge a MAC it could ensure that Π_I^s and Π_J^t are *never* partners.

Let ν_e be the probability that $\mathcal{A}$ succeeds in winning its security game without forging a MAC. The probability that $\mathcal{ME}$ chooses the right test session is $\frac{1}{n_p^2 n_s}$. Hence the advantage of $\mathcal{ME}$ in guessing correctly is:

$$\text{Succ}^{\mathcal{ME}} \geq \frac{\nu_e}{n_p^2 n_s}.$$

Conclusion of proof

The proof concludes by observing that:

$$\begin{aligned} \mathrm{Succ}^{\mathcal{A}} &\leq \nu_f + \nu_e \\ &\leq n_p \mathrm{Succ}^{\mathcal{F}} + n_p^2 n_s \mathrm{Succ}^{\mathcal{ME}} \end{aligned}$$

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.

Extending the B–R model

Initial work of Bellare and Rogaway in 1993 was followed up by many others:

- 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)

The Canetti–Krawczyk model

- Stems from two papers by Bellare–Canetti–Krawczyk (1998) and Canetti–Krawczyk (2001)
- Allows use of 'authenticators' for design of protocols
- Security definition similar to Bellare and Rogaway, but:
  - session identifiers chosen by environment
  - allows use of session key to provide secure channels
- HMQV was proven secure by Krawczyk in 2005 using a variant of CK model.

The eCK model

- Extended Canetti–Krawczyk model due to LaMacchia, Lauter and Mityagin (2007)
- Allows adversary to obtain anything that does not 'trivially' reveal session key
- Only applicable to two-party two-pass key agreement protocols

Some unresolved issues

- So far there are relatively few protocols with proofs in the standard model (not using random oracles).
- Models for group protocols are more complex: insider attacks for group models need to be carefully defined.
- Is it better to provide proofs in the universally composability (UC) model?