6.897: Selected Topics in Cryptography
Lectures 15 and 16

Lecturer: Ran Canetti

Highlights of last week's lectures

- Universal composition with joint state: Allows analyzing a multi-instance system as separate instances, even when the instances use a joint "subroutine". Examples:
  - Protocols in the F_{crs}-hybrid model
  - Protocols that use signature schemes
- UC formulation of signature schemes:
  - Motivation for providing a UC formulation of signatures
  - The signature functionality, F_{sig}.
  - Equivalence with CMA-security
- Achieving authenticated communication:
  - Defined F_{cert} (I.e, F_{sig} with binding to a party rather than a verif. key)
  - Realized F_{cert} given F_{sig} and public registries (F_{sig}).
  - Realized F_{auth} given F_{cert}.

This week:

- Authenticated Key Exchange and Secure-Session protocols:
  - Brief history of the problem and formalizations;
  - UC formalization of KE: The F_{ke} functionality.
  - Signature-based KE: Realizing F_{ke} in the F_{cert}–hybrid model
  - From KE to secure channels:
    - The secure session functionality F_{ss}.
    - Realizing F_{ss} in the F_{ke}–hybrid model.
  - Further relaxation of F_{ke}:
    - Non-information oracles
    - Equivalence with an earlier definition.
- UC formulation of public-key encryption:
  - Motivation for providing a UC formulation of encryption
  - The public-key encryption functionality, F_{pke}.
  - Equivalence with CCA2-security for non-adaptive adversaries
  - Problems and solutions for adaptive adversaries
  - Relaxing CCA security

Background on key exchange protocols

- Key Exchange: A protocol for two parties to generate a common random key that is “secret” for external adversaries. Variants:
  - Assuming authenticated communication (the Diffie-Hellman model)
  - Unauthenticated communication (“Authenticated Key Exchange”)
- Arguably the most widely used cryptographic protocol. Typical use:
  - Run an AKE protocol to agree on a common secret
  - Derive keys for symmetric encryption and authentication functions
  - Use the derived keys to encrypt/authenticate the communication within the session.
- Different ways to authenticate the exchange:
  - Long-term public keys for signature or encryption, plus “public-key infrastructure”.
  - Long-term pre-shared keys
  - Trusted third parties (The Kerberos model)
  - Passwords

Analyses of key exchange protocols

- AKE has been studied extensively but remained evasive:
  - Protocols were proposed, standardized, and later broken
  - Analytical approaches were proposed and later broken (e.g. the Needham-Schroeder78 protocol was later proven secure in the Burrows-Abadi-Needham89 logic, only to be broken in Lowe95)…
- First complexity-based notion of security by Bellare-Rogaway93:
  - Based on a “distinguishing game” for the adversary
  - Explicitly handles multiple concurrent sessions
- A bug and a fix by Rackoff (circa 95)
- Treatments that argue usability for secure sessions:
  - Bellare-C-Krawczyk98: simulation based (but has problems)
  - Shoup99: Points to problems in BCK98 and some fixes.
  - C-Krawczyk01: based on BR93 with a different system model, defines and obtains “secure sessions”.
  - CK02: A UC treatment of AKE

The CK01 notion (based on BR93)

Consider an adversary A that interacts with a set of parties:

- Ideal initialization: The parties obtain secret keys and the public keys of everyone else for a generation function specified in the protocol.
- A can activate a party Pi with input (sid,Pj)
  (I.e., to exchange a new key with Pj and session ID sid)
- A obtains all messages sent by the parties and delivers arbitrary messages (models unauthenticated network).
- A can corrupt parties and obtain their local data (either the session data or the long-term key or both).
- A can ask to “reveal” a session (sid,Pi); in response it gets the local output of (sid,Pi), which is of the form (sid,Pj,a) for a in $\{0,1\}^k$.
- At some point, A chooses a “test session” (sid*,Pj*) within Pi* and receives a “test value” a*, where a* is either taken from Pi*’s output or is randomly chosen.
- Later, A outputs a guess whether a* was taken from Pi*’s output. It “wins” if neither the session (sid*,Pj*) within Pi* nor the session (sid*,Pi*) within Pj* are corrupted or revealed, and the guess is correct.

A protocol is “SK-secure” if for any A:

Agreement: Whenever Pi,Pj output (sid,Pj,a) and (sid,Pi,a’) we have a = a’.

Secrecy: A wins the game with probability only negligibly more than ½.

More on the CK01 notion

In addition, they:

- Define a notion of secure (secret and authenticated) sessions, along the same lines.
- Prove that the composition of
  - A secure KE protocol
  - Key derivation using pseudorandom functions
  - Standard symmetric encryption and MAC using the derived keys

  Is a good secure-session protocol.

Caveats:

- No concrete assurance in the adequacy of the notion of secure-sessions.
- Multiple sessions have to be explicitly treated in the definition.
- There is no security guarantee w.r.t. other protocols
- Only idealized set-up.

A UC treatment of key exchange

- Define and analyze key exchange protocols for a single session.
  - Use the JUC theorem to deduce security in the multi-session case.
  - Use the UC theorem to deduce security with respect to other protocols.
- Define a UC notion of a secure session (again, for a single session) and show how to realize it given UCKE.

- Question: How does the new definition relate to the old one?

The key-exchange functionality F_{KE} (I)

Wait to receive:

- (sid,Pi,Pj) from party (sid,Pi)
- (sid,Pj,Pi) from party (sid,Pj)

Then:

- Choose a $\leftarrow_R \{0,1\}^k$
- Output (sid,Pi,Pj,a) to (sid,Pi) and (sid,Pj)
- Send (sid,Pi,Pj) to the adv.
- Halt.

Too strong: Forces a to be random even if one of the parties is corrupted.

The key-exchange functionality F_{KE} (II)

Wait to receive:

- (sid,Pi,Pj) from party (sid,Pi)
- (sid,Pj,Pi) from party (sid,Pj)

Then:

- If one of the parties is corrupted then obtain a value a from the adv. Else, choose a $\leftarrow_R \{0,1\}^k$
- Output (sid,Pi,Pj,a) to (sid,Pi) and (sid,Pj)
- Send (sid,Pi,Pj) to the adv.
- Halt.

Too strong: Requires “mutual authentication”

The key-exchange functionality F_{KE} (III)

When receiving (sid,Pi,Pj) from the first party, (sid,Pi), do:

- Send (sid,Pi,Pj) to the adv, obtain (“ok”,Pi) from the adv.
- If one of the parties is corrupted then obtain a value a from the adv. Else, choose a $\leftarrow_R \{0,1\}^k$
- Output (sid,Pi,Pj,a) to (sid,Pi)

When receiving (sid,Pj,Pi) from the second party, (sid,Pj), do:

- Send (sid,Pi,Pj) to the adv, obtain (“ok”,Pj) from the adv.
- Output (sid,Pi,Pj,a) to (sid,Pi).

Signature-based key exchange

.

- Realize F_{ke} in the F_{cert}–hybrid model (here each session uses its own copy of F_{cert}).

- Recall the protocol for realizing FF_{cert} using a single copy of F_{cert} (essentially, sign the ssid together with the message).

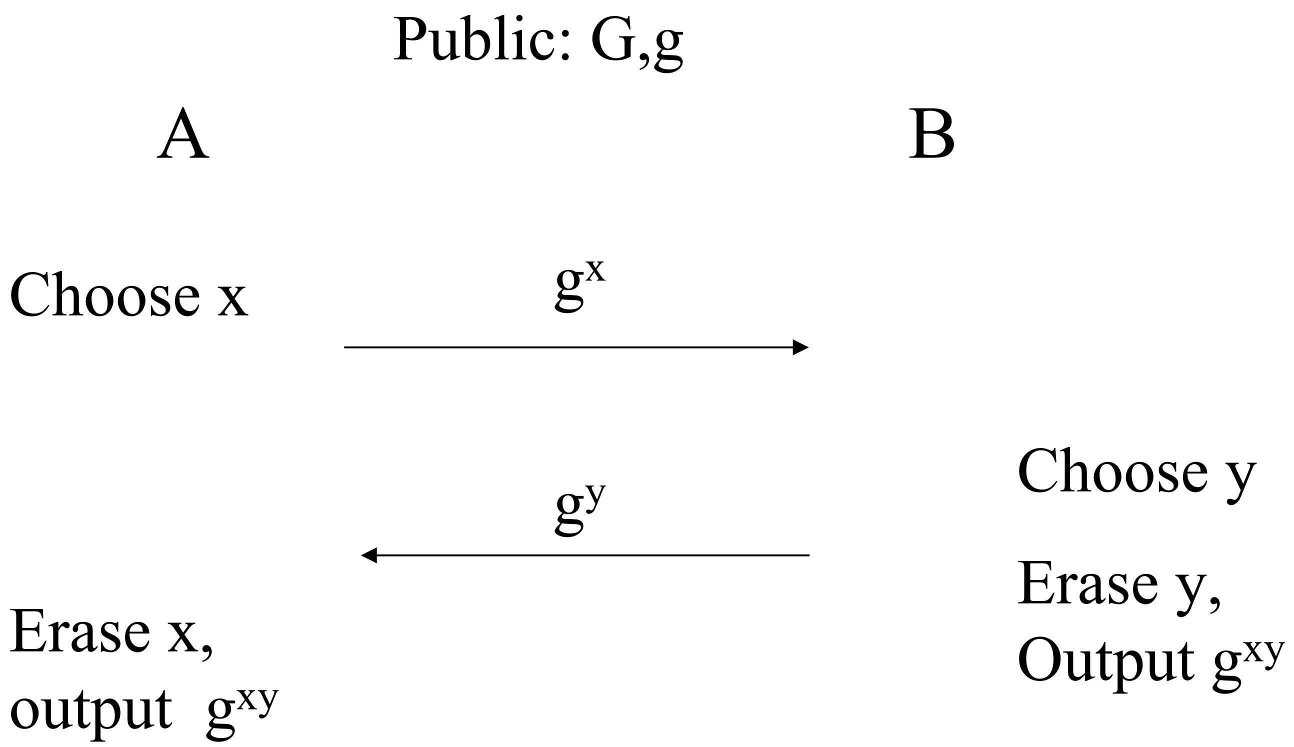
- Use the JUC theorem to obtain a multi-instance key-exchange protocol where each party uses a single signing/verification key pair.

Reminder: The certification functionality, F_{cert}

1. On input (sid,“sign”,m) from (sid,S), where sid=(S,sid’), forward (sid,m) to A, obtain a “signature” s from A, output s to (sid,S), and record (m,s,1). Verify that no prior record (m,s,0) exists.
2. On input (sid,“verify”,m,s) from any party, return (sid,f) where:
   - If (m,s,b) is recorded then f=b.
   - If S is uncorrupted and (m,s*,1) is not recorded for any s*, then f=0.
   - Else, forward (m,s) to A, obtain f from A, and record (m,s,f).

F_{cert} is similar to F_{sig} except that the KeyGen interface is deleted. Instead, verification is done directly with respect to the signer’s identity (which appears in the sid).

Basic (unauthenticated) Diffie-Hellman



-This protocol guarantees secrecy of the key against eavesdroppers,
under the Decisional Diffie-Hellman Assumption.
-More abstractly, can be based on any semantically secure encryption scheme.

Authenticated Diffie-Hellman: Attempt I

Sign the exchange:

$$\begin{tikzcd}
A & & & B \\
& {} \arrow[r, "g^x"] & {} & \\
& {} & {} \arrow[l, "{g^y,\ \mathrm{SIG}_B(g^x,g^y)}"'] & \\
& {} \arrow[r, "{\mathrm{SIG}_A(g^x,g^y)}"] & {} &
\end{tikzcd}$$

Attack on Attempt I:

$$\begin{tikzcd}[column sep=huge]
A & E & B \\
{} \arrow[r, "g^x"] & {} \arrow[r, "g^x"] & {} \\
{} & {} \arrow[l, "{g^y, SIG_B(g^x,g^y)}"'] & {} \arrow[l, "{g^y, SIG_B(g^x,g^y)}"'] \\
{} \arrow[r, "{SIG_A(g^x,g^y)}"] & {} \arrow[r, "{SIG_E(g^x,g^y)}"] & {}
\end{tikzcd}$$

Attack on Attempt I:

$$\begin{tikzcd}[column sep=huge]
A & E & B \\
{} \arrow[r, "g^x"] & {} \arrow[r, "g^x"] & {} \\
{} & {} \arrow[l, "{g^y, SIG_B(g^x,g^y)}"'] & {} \arrow[l, "{g^y, SIG_B(g^x,g^y)}"'] \\
{} \arrow[r, "{SIG_A(g^x,g^y)}"] & {} \arrow[r, "{SIG_E(g^x,g^y)}"] & {} \\
{} \arrow[rr, "\text{“Please transfer a million dollars to my account”}"] & & {}
\end{tikzcd}$$

Authenticated Diffie-Hellman: Attempt II

Idea: Include the identities in the signed text.

$$\begin{tikzcd}
A & & B \\
{} \arrow[rr, "{g^x , A}"] & & {} \\
{} & & {} \arrow[ll, "{g^y, B, SIG_B(g^x,g^y,A,B)}"'] \\
{} \arrow[rr, "{SIG_A(g^x,g^y,A,B)}"] & & {}
\end{tikzcd}$$

Authenticated Diffie-Hellman: Attempt II

Idea: Include the identities in the signed text.

$$\begin{tikzcd}
A & & B \\
{} \arrow[rr, "{g^x , A}"] & & {} \\
{} & & {} \arrow[ll, "{g^y, B, SIG_B(g^x,g^y,A,B)}"'] \\
{} \arrow[rr, "{SIG_A(g^x,g^y,A,B)}"] & & {}
\end{tikzcd}$$

This is essentially the ISO 9798-3 protocol.

Theorem: The above protocol securely realizes F_{ke} in the F_{cert}–hybrid model.

Drawbacks of the protocol:

- Identities of A,B are transmitted in the clear (no identity protection).
- Leaves a non-repudiable proof of the exchange, including identities.
- Including the peer identity in the signature is harder to implement and introduces latency.

Can we do better?

Another approach: The SIGMA protocol [Krawczyk 95] (based on ideas from the STS protocol of [Diffie-vanOorschot-Wiener92])

Basic version:

g^{xy}=k0,k1
K0 is the session key

$$\begin{tikzcd}[column sep=huge]
A & B \\
{} \arrow[r, "g^x"] & {} \\
{} & {} \arrow[l, "{g^y, B, SIG_B(g^x,g^y), MAC_{k1}(0,B)}"'] \\
{} \arrow[r, "{A, SIG_A(g^x,g^y), MAC_{k1}(1,A)}"] & {}
\end{tikzcd}$$

The SIGMA protocol: Encrypted version

g^{xy}=k0,k1,k2
K0 is the session key

$$\begin{tikzcd}[column sep=huge] A & B \\ {} \arrow[r, "g^x"] & {} \\ {} & {} \arrow[l, "C=ENC_{k2}(g^y, B, SIG_B(g^x,g^y)), PRF_{k1}(C)"'] \\ {} \arrow[r, "D= ENC_{k2}(A, SIG_A(g^x,g^y)), PRF_{k1}(D)"] & {} \end{tikzcd}$$

- Provides identity protection from eavesdroppers for both parties
- “Full identity protection” for the initiator.

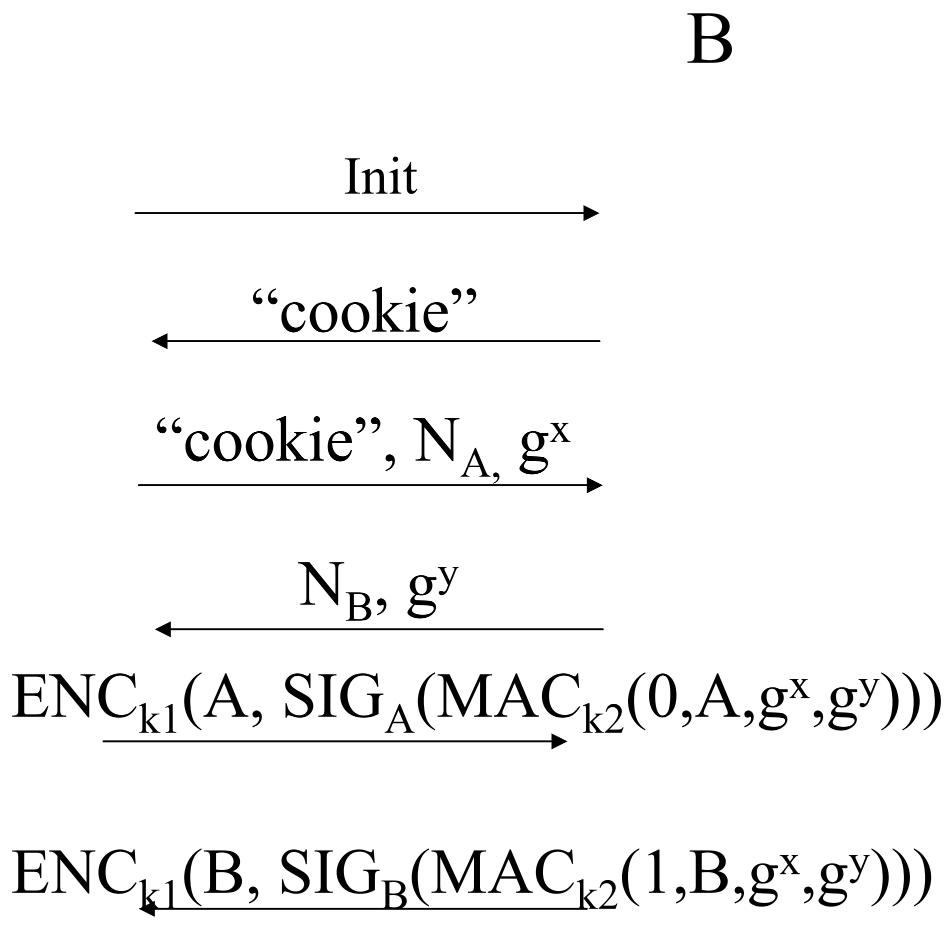
Theorem: The SIGMA protocol securely realizes F_{ke} in the F_{cert}–hybrid model.

Note: While the “ISO protocol” provides mutual authentication, SIGMA does not.

IPSEC's IKE protocol ("crypto core"):

K0,k1,k2 are
Derived from g^{xy}.

Notes:
- Cookies for DOS protection
- Full ID protection for responder


A
B
Init
"cookie"
"cookie", N_{A}, g^x
N_B, g^y
$ENC_{k1}(A, SIG_A(MAC_{k2}(0,A,g^x,g^y)))$
$ENC_{k1}(B, SIG_B(MAC_{k2}(1,B,g^x,g^y)))$

The secure session functionality, F_{ss}

1. When receiving input(sid,"init",Pi,Pj) from (sid,Pi), record Pi and Pj as the peers of this session and send (sid,Pi,Pj) to the adv.
2. When receiving input (sid,m) from a peer (sid,Pi), do:
    1. Output (sid,m) to the other peer
    2. Send (sid,Pi,|m|) to the adv.

Realizing F_{ss} in the F_{ke}-hybrid model

Primitives used:

- a pseudorandom function PRF
- a semantically secure symmetric encryption scheme (ENC,DEC)
- a symmetric message authentication function MAC

Protocol:

- On input (sid,"init",Pi,Pj), (sid,Pi) does:
  - Calls F_{ke} with (sid.0,Pi,Pj), obtains key k.
  - Lets k0=$PRF_k(0)$,…, k3=$PRF_k(3)$.
  - Initializes counters IN and OUT to 0.
- On input (sid,m), compute C=$ENC_{ko}(m)$, and send (sid,C,MAC_{k1}(C,OUT++)).
- When receiving (sid,C,D), verify that D=MAC_{k3}(C,IN++)), and if so then outputs (sid, DEC_{k3} (C)).

Remaining Questions:

- What’s the relation between the UC notion of KE and the prior one (SK-security)?
- Why does adaptive security require special encryption schemes?

- Restrict the discussion to “session-wise protocols”, ie protocols where the only shared state across sessions is the long-term authentication module.

Theorem: If a protocol realizes F_{ke} then it is SK-secure.

The other direction doesn’t hold…

Example: DH exchange (assuming authenticated channels)

Protocol 2DH

A   B

Choose x $\xrightarrow{\quad g^x \quad}$

              Choose y

$\xleftarrow{\quad g^y \quad}$

              Erase y,
              Output g^{xy},

Erase x,
output g^{xy}

Proof of security
(based on DDH assumption):

Assume an adversary *A* that breaks 2DH. Construct a distinguisher of (g^x,g^y,g^{xy}) from (g^x,g^y,g^z).

Given (*a,b,c*) do:

- Choose *i* at random. (Hope that the *i*th session will be the test session. If not, then output a random bit.)
- Run *A*. In the *i*th session, give *A* the values *a,b* as the messages sent by the parties, and *c* as the test value.
- Output whatever *A* outputs.

Note: A never expects to see the secret exponents, since it never corrupts the test session.

2DH does not realize F_{ke}

Z that distinguishes between real and ideal:

- Prompt P1,P2 to exchange a key.
- Obtain P1's message, *a,* from *A.*
- Obtain P2's message, *b,* from *A.*
- Before P2's message is delivered to P1:
  - Obtain the session key *k* from P2's output.
  - Instruct *A* to corrupt P1, obtain the secret exponent *x*.
- Output "real" if $g^x=a$ *and* $b^x=k$.

Analysis:

- *In a real execution, Z always outputs "real".*
- *In the ideal process k is independent of a,b. Thus Z outputs "real" w.p. 1/|group|.*

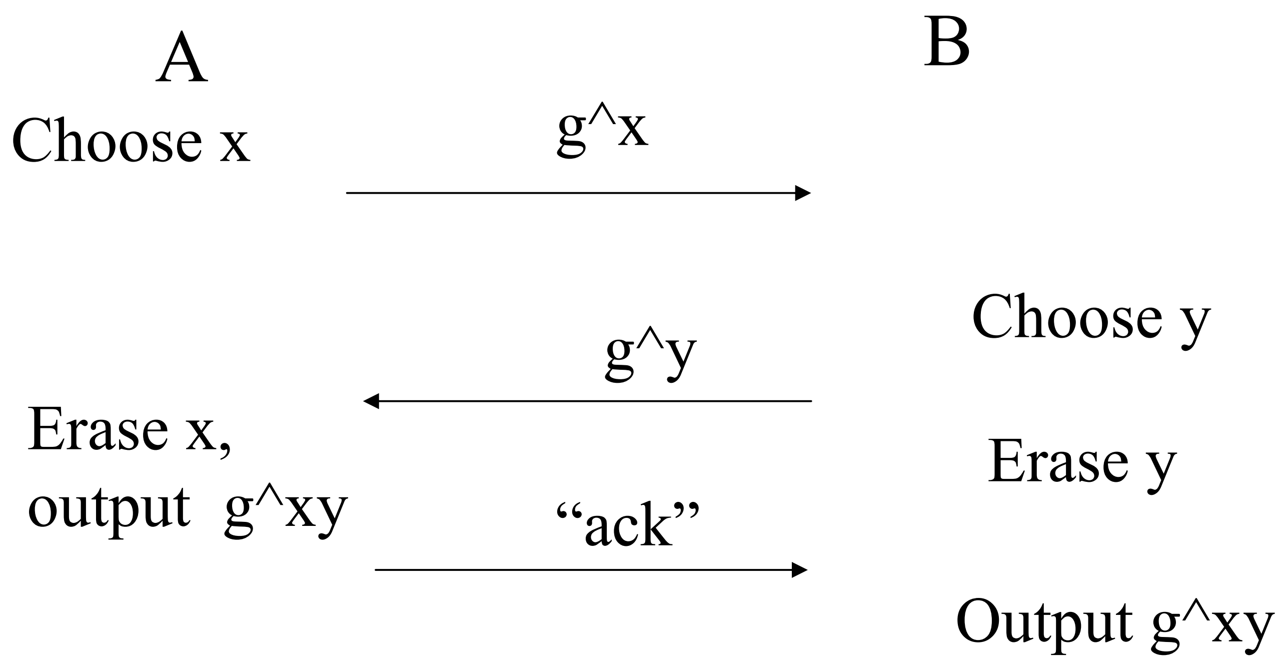
Reflections

The essence of the problem: P2 outputs the key while P1 still holds “sensitive info”.

- Is this a “real” security problem with 2DH?
  Or is it only a “technicality” of the definition?

- Are there reasonable ways to strengthen 2DH, or alternatively to relax the definition?

Solution 1: An addition to 2DH:



-*Can show that a similar addition turns* ***any*** *SK-secure protocol into a UC-secure one.*

Solution 2: Relax F_{KE}

The idea: Allow F_{KE} to “release information” on the secret key, as long as this information is indistinguishable from random. That is:

- An ITM M is a ***non-information oracle*** if no adversary, after interacting with M, can distinguish the local output of M from a random value.
- Modify F_{KE} as follows:
  - F_{KE} will allow the adversary to interact with a NIO M.
  - The session key will be the local output of M.
  - When either party is corrupted, the adversary gets the internal randomness of M.

The Weak key-exchange functionality F_{wkE} (with non-information oracle M)

When receiving (sid,Pi,Pj) from the first party, (sid,Pi), do:

- Send (sid,Pi,Pj) to the adv.
- Let the adv. interact with a copy Mi of M.
- If Pi is corrupted then reveal the local state of Mi to the adv, and let the adv. set the output of Mi.

When receiving (sid,Pj,Pi) from the second party, (sid,Pj), do:

- Send (sid,Pj,Pi) to the adv.
- Let the adv. interact with a copy Mj of M.
- If Pj is corrupted then reveal the local state of Mj to the adv, and let the adv. set the output of Mj.

When either Mi or Mj generate output a, output (sid,Pi,Pj,a) to the corresponding party. When the other copy of M generates output, output (sid,Pi,Pj,a) to the other party.

Theorem: A KE protocol P is SK-secure iff there exists a non-information oracle such that P realizes F_{wke}^{M}.

Theorem: Any protocol that securely realizes F_{ss} in the F_{ke}-hybrid model, realizes F_{ss} also in the F_{wke}^{M}-hybrid model for any NIO M.

A similar trick works for the “adaptive encryption” problem in realizing F_{ss}:

- Relax F_{ss} by adding a “non-information oracle” for encryption.
- Can show that any semantically secure encryption can be used to realize F_{wss} for adaptive adversaries.

Modeling public-key encryption as an ideal functionality

Same motivation as for signatures:

- Re-assert validity of known notions
- Facilitate modular analysis of protocols using PKE (e.g.,via the JUC theorem)
- A step towards formal/automated analysis of protocols.

Functionality F_{pke} (I) (parameterized by domain M)

On input (sid, KeyGen) from D, do:

- Verify that sid=(D,sid')
- Hand (KeyGen, sid) to adv, get value e.
- Return e to D.

On input (sid,Encrypt,e',m) from any P, do:

- If m not in M then return an error message
- Else, hand (Encrypt,sid,|m|) to adv.
  (If e' !=e then give the full m to the adv.)
- Get c from adv, record (m,c) and return c to P.

On input (sid,Decrypt,c) from D (and D only), do:

- If a pair (m,c) is recorded, then return m to D_i.
- Else, hand c to adv, get value m, and return m to D.

Equivalence with CCA security

Given an encryption scheme E=(Gen, Enc, Dec), construct the protocol P_E:

- On input (sid,KeyGen), verify that sid=(D,sid') where D is the local pid; then run (e,d)$\leftarrow$Gen(), return e and record d.
- On input (sid,Encrypt,m,e), return Enc_e(m).
- On input (sid,Decrypt,c), D retrieves d and returns Dec_d(c).

Theorem:

An encryption scheme E is CCA2-secure iff protocol P_E securely realizes F_{pke} for non-adaptive adversaries.

Reminder: CCA2 Security

An encryption scheme (Gen, Enc, Dec) is CCA2 secure for domain D if:

- Validity: For all m in D: $Dec_d(Enc_e(m))=m$
- CCA2 security:
  - Define game for adversary A:
    - $(e,d) \leftarrow G(k)$; $A \leftarrow e$
    - $c \leftarrow A$; $A \leftarrow D_d(c)$
    - $(m_0,m_1) \leftarrow A$; $A \leftarrow c^* = E_e(m_b)$, $b \leftarrow_R \{0,1\}$
    - $c \leftarrow A$; $A \leftarrow D_d(c)$, unless $c=c^*$
    - $b' \leftarrow A$
  - Scheme secure if any A outputs b with prob. $< \frac{1}{2}$ + negl.

Proof of equivalence:

P_E realizes F_{pke} ➔ E is CCA2-secure:

Validity: Assume E is not valid, then construct an environment Z and adversary A that distinguish a run of P_E from the ideal process for F_{pke}: Z invokes a simple KeyGen→Encrypt→ Decrypt sequence for an uncorrupted decryptor.

CCA2 security: Assume there exists a breaker B for E. Z runs B:

- Z Invokes an uncorrupted D with KeyGen, obtains e, gives to B.
- When B asks to encrypt (m_0,m_1), Z chooses $b\leftarrow\{0,1\}$, asks D to encrypt m_b, obtains c*, gives c* to B as test ciphertext.
- When B asks to decrypt c, Z asks D to decrypt c.
- When B outputs a guess b', Z outputs b+b'.

Analysis: In a run of P^H, Z outputs 1 with non-neglig. probability. In the ideal process, Z never outputs 1.

Analysis of Z:

- If Z interacts with P_E then the view of the simulated B sees is exactly that of an interaction with E. Thus, if B has advantage f then Z outputs 1 w.p. ½+f.

- If Z interacts with some S in the ideal process for F_{pke} then the view of B is independent from b. (This is so since B sees only e and the ciphertexts and decryptions generated by S, and the view of S is also independent of b.) Thus Z outputs 1 w.p. ½.

E is CCA2-secure ➔ P_E realizes F_{pke}:

Let Z be an environment that distinguishes a run of P_E from ideal interaction with F_{pke} w.p. f for any ideal-process adversary S. In particular, Z works for the following "generic S":

- When asked by F_{pke} to generate a key, S runs (e,d)$\leftarrow$Gen() and returns e.
- When asked by F_{pke} to generate a ciphertext, S runs c$\leftarrow$Enc(e,$0^{|m|}$) and returns c.
- When asked by F_{pke} to decrypt(c), S returns m$\leftarrow$Dec(d,c).

Let n be the number times Z asks to encrypt a message. Define n+1 hybrid interactions $H_0 \ldots H_n$:

In H_i the first i ciphertexts are computed as Enc(e,$0^{|m|}$) , and the rest are computed as Enc(e,m) (where m is the plaintext in the request).

Then there is an $i<n$ s.t. Z distinguishes between H_i and H_{i+1} with probability f/n.

Given Z, contruct a breaker B for E. B runs Z:

- When Z activates the signer with KeyGen, B gives Z the e from B's input.
- In the first i times that Z asks a party to encrypt a message m, B returns Enc(e,$0^{|m|}$).
- In the i+1st time that Z asks a party to encrypt a message m, B gives (m, $0^{|m|}$) to its Enc(e,$0^{|m|}$). encryption oracle, gets c*, and gives c* to Z.
- In the rest of the times that Z asks a party to encrypt a message m, B returns Enc(e,m).
- When Z asks D to decrypt a ciphertext c that was generated by B, then B returns the corresponding m. If c was not generated by B, then B consults its decryption oracle and returns the answer to Z. (B never asks to decrypt c*).

Claim: If c* is an an encryption of m then Z sees H_i . If c* is an encryption of $0^{|m|}$ then Z sees H_{i+1}.

■

Note: If Z asks to corrupt D during the interaction then F is stuck…

A definitional caveat:

F_{pke} allows an “illegal ciphertext” to be decrypted to a value that became known only after the ciphertext was generated.

This seems to allow “functional malleability”…

Q: Is that a contradiction to the equivalence with CCA2-security?

A: No… since the equivalence was proven only to encryption schemes where the decryption algorithm does not get external information other than the ciphertext (e.g., network messages), thus the problem does not arise.

Q: How to guarantee this property without so restricting the decryption algorithm?

Functionality F_{pke} (II)
(parameterized by domain M)

On input (sid, KeyGen) from D, do:

- Verify that sid=(D,sid')
- Hand (KeyGen, sid) to adv, get e and algorithms Enc,Dec.
- Return e to D.

On input (sid,Encrypt,e',m) from any P, do:

- If m not in M then return an error message
- Else, hand (Encrypt,sid,|m|) to adv.
  (If e' !=e then give the full m to the adv.)
- Compute c=E(m), record (m,c) and return c to P.

On input (sid,Decrypt,c) from D (and D only), do:

- If a pair (m,c) is recorded, then return m to D_i.
- Else, compute m=D(c) and return m to D.

Properties of the new formulation

- The decryption value of a ciphertext is not influenced by events occurring after the ciphertext was generated, regardless of how the decryption procedure works.
- Equivalence with CCA2-security still works.

Summary of course up till now

- Saw two frameworks for analyzing security of cryptographic protocols:
  - “basic security”: Provides the basic ideas, but limited scope (synchronous, non-reactive), limited composability (non-concurrent).
  - “UC security”: More general scope, general composability, but more restrictive.
- Saw general construction and proof techniques (“how to realize any ideal funtionality”) within the UC security framework.
- Saw an imossibility result (there are more…)
- Saw how to model common tasks within the UC framework (signatures, authentication, key-exchange, encryption…)

Things to keep in mind

- Composability is an integral part of a security requirement.
- Notions of security are not set in stone. (Deciding what theorems to prove on a protocol is sometimes harder than actually proving…)
- Formulating ideal functionalities “right” is very tricky.
- There is usually no such thing as “THE right notion of security”. Different notions have different properties and are good for different purposes.