Cryptographically sound formal verification of security protocols

Two views of cryptography

Formal ("Dolev-Yao") view

- Messages — elements of a term algebra.
- Possible operations on messages are enumerated.
- Choices in semantics — non-deterministic.
  - Protocol and the adversary are easily represented in some process calculus.

Computational view

- Messages — bit strings.
- Possible operations on messages — everything in PPT.
- Choices in semantics — probabilistic.
  - Protocol and adversary — a set of probabilistic interactive Turing machines.

Two views of cryptography

Formal ("Dolev-Yao") view

- Messages — elements of a term algebra.
- Possible operations on messages are enumerated.
- Choices in semantics — non-deterministic.
  - Protocol and the adversary are easily represented in some process calculus.
- Simpler to analyse.

Computational view

- Messages — bit strings.
- Possible operations on messages — everything in PPT.
- Choices in semantics — probabilistic.
  - Protocol and adversary — a set of probabilistic interactive Turing machines.
- Closer to the real world.

Table of Contents

- The Abadi-Rogaway result on the indistinguishability of computational interpretations of formal messages.
- Translating protocol traces between formal and computational world.

A simple language for messages

The atomic building blocks:

- Formal keys $k, k_1, k_2, k', k'', \ldots \in \mathbf{Keys}$
- Formal coins $r, r_1, r_2, r', r'', \ldots \in \mathbf{Coins}$
- Bits $b \in \{0, 1\}$

A simple language for messages

The atomic building blocks:

- Formal keys $k, k_1, k_2, k', k'', \ldots \in \mathbf{Keys}$
- Formal coins $r, r_1, r_2, r', r'', \ldots \in \mathbf{Coins}$
- Bits $b \in \{0, 1\}$

A formal expression $e \in \mathbf{Exp}$ is

$$\begin{array}{rcl} e & ::= & k \\ & | & b \\ & | & (e_1, e_2) \\ & | & \{e'\}_k^r \end{array}$$

If $\{e\}_k^r$ and $\{e'\}_{k'}^r$ both occur in an expression then $k = k'$ and $e = e'$.

A simple language for messages

The atomic building blocks:

- Formal keys $k, k_1, k_2, k', k'', \ldots \in \mathbf{Keys}$
- Formal coins $r, r_1, r_2, r', r'', \ldots \in \mathbf{Coins}$
- Bits $b \in \{0, 1\}$

A formal expression $e \in \mathbf{Exp}$ is

$$\begin{array}{rcl} e & ::= & k \\ & | & b \\ & | & (e_1, e_2) \\ & | & \{e'\}_k^r \end{array}$$

If $\{e\}_k^r$ and $\{e'\}_{k'}^r$ both occur in an expression then $k = k'$ and $e = e'$.

- e is similar to Dolev-Yao messages.

A simple language for messages

The atomic building blocks:

- Formal keys $k, k_1, k_2, k', k'', \ldots \in \mathbf{Keys}$
- Formal coins $r, r_1, r_2, r', r'', \ldots \in \mathbf{Coins}$
- Bits $b \in \{0, 1\}$

A formal expression $e \in \mathbf{Exp}$ is

$$\begin{array}{rcl} e & ::= & k \\ & | & b \\ & | & (e_1, e_2) \\ & | & \{e'\}_k^r \end{array}$$

If $\{e\}_k^r$ and $\{e'\}_{k'}^r$ both occur in an expression then $k = k'$ and $e = e'$.

- e is similar to Dolev-Yao messages.
- We can also interpret it as a program for computing a message.

Semantics — building blocks

- Let $\langle \cdot, \cdot \rangle : (\{0,1\}^*)^2 \to \{0,1\}^*$ be easily computable and invertible injective function.

Semantics — building blocks

- Let $\langle \cdot, \cdot \rangle : (\{0,1\}^*)^2 \to \{0,1\}^*$ be easily computable and invertible injective function.
- A symmetric encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$:
  - $\mathcal{K}\ (1^\eta)$ — generates keys;
  - $\mathcal{E}\ (1^\eta, \mathtt{k}, \mathtt{x})$ — encrypts $\mathtt{x}$ with $\mathtt{k}$;
  - $\mathcal{D}(1^\eta, \mathtt{k}, \mathtt{y})$ — decrypts $\mathtt{y}$ with $\mathtt{k}$.

  $\mathcal{K}$ and $\mathcal{E}$ — probabilistic, $\mathcal{D}$ — deterministic.

Semantics — building blocks

- Let $\langle \cdot, \cdot \rangle : (\{0,1\}^*)^2 \to \{0,1\}^*$ be easily computable and invertible injective function.
- A symmetric encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$:
  - $\mathcal{K}^{\mathtt{r}}(1^\eta)$ — generates keys from random coins $\mathtt{r}$;
  - $\mathcal{E}^{\mathtt{r}}(1^\eta, \mathtt{k}, \mathtt{x})$ — encrypts $\mathtt{x}$ with $\mathtt{k}$ using the random coins $\mathtt{r}$;
  - $\mathcal{D}(1^\eta, \mathtt{k}, \mathtt{y})$ — decrypts $\mathtt{y}$ with $\mathtt{k}$.

  $\mathcal{K}$ and $\mathcal{E}$ — probabilistic, $\mathcal{D}$ — deterministic.

Semantics — building blocks

- Let $\langle\cdot,\cdot\rangle : (\{0,1\}^*)^2 \to \{0,1\}^*$ be easily computable and invertible injective function.
- A symmetric encryption scheme $(\mathcal{K},\mathcal{E},\mathcal{D})$:
  - $\mathcal{K}^{\mathtt{r}}(1^\eta)$ — generates keys from random coins $\mathtt{r}$;
  - $\mathcal{E}^{\mathtt{r}}(1^\eta,\mathtt{k},\mathtt{x})$ — encrypts $\mathtt{x}$ with $\mathtt{k}$ using the random coins $\mathtt{r}$;
  - $\mathcal{D}(1^\eta,\mathtt{k},\mathtt{y})$ — decrypts $\mathtt{y}$ with $\mathtt{k}$.

  $\mathcal{K}$ and $\mathcal{E}$ — probabilistic, $\mathcal{D}$ — deterministic.

  Correctness:

$$\forall \eta,\mathtt{x},\mathtt{r},\mathtt{r}' : \begin{array}{l} \mathtt{k} := \mathcal{K}^{\mathtt{r}}(\eta) \\ \mathtt{y} := \mathcal{E}^{\mathtt{r}'}(\eta,\mathtt{k},\mathtt{x}) \\ \mathtt{x}' := \mathcal{D}(\eta,\mathtt{k},\mathtt{y}) \\ (\mathtt{x}=\mathtt{x}')? \end{array}$$

Semantics of a formal expression

- For each $k \in \mathbf{Keys}$ let $\mathtt{s}_k \leftarrow \mathcal{K}(1^\eta)$
- For each $r \in \mathbf{Coins}$ let $\mathtt{s}_r \in_R \{0,1\}^\omega$.

Define

$$\begin{aligned}
[\![k]\!]_\eta &= \mathtt{s}_k \\
[\![b]\!]_\eta &= b \\
[\![(e_1, e_2)]\!]_\eta &= \langle [\![e_1]\!]_\eta, [\![e_2]\!]_\eta \rangle \\
[\![\{e'\}_k^r]\!]_\eta &= \mathcal{E}^{\mathtt{s}_r}(\eta, \mathtt{s}_k, [\![e']\!]_\eta)
\end{aligned}$$

Semantics of a formal expression

- For each $k \in \mathbf{Keys}$ let $\mathtt{s}_k \leftarrow \mathcal{K}(1^\eta)$
- For each $r \in \mathbf{Coins}$ let $\mathtt{s}_r \in_R \{0,1\}^\omega$.

Define

$$
\begin{aligned}
[\![k]\!]_\eta &= \mathtt{s}_k \\
[\![b]\!]_\eta &= b \\
[\![(e_1, e_2)]\!]_\eta &= \langle [\![e_1]\!]_\eta, [\![e_2]\!]_\eta \rangle \\
[\![\{e'\}_k^r]\!]_\eta &= \mathcal{E}^{\mathtt{s}_r}(\eta, \mathtt{s}_k, [\![e']\!]_\eta)
\end{aligned}
$$

$[\![\cdot]\!]$ assigns to each formal expression a family of probability distributions over bit-strings

Computational indistinguishability

We are looking for sufficient conditions in terms of e_1 and e_2 for

$$[\![e_1]\!] \approx [\![e_2]\!] \ .$$

Computational indistinguishability

We are looking for sufficient conditions in terms of e_1 and e_2 for

$$[\![e_1]\!] \approx [\![e_2]\!] \ .$$

Two families of probability distributions over bit-strings $D^0 = \{D^0_\eta\}_{\eta\in\mathbb{N}}$ and $D^1 = \{D^1_\eta\}_{\eta\in\mathbb{N}}$ are computationally indistinguishable if for all PPT algorithms $\mathcal{A}$:

$$\Pr[b = b^* \mid b \in_R \{0,1\}, x \leftarrow D^b_\eta, b^* \leftarrow \mathcal{A}(1^\eta, x)] = 1/2 + \varepsilon(\eta)$$

for some negligible function ε.

Computational indistinguishability

We are looking for sufficient conditions in terms of e_1 and e_2 for

$$[\![e_1]\!] \approx [\![e_2]\!] \ .$$

Two families of probability distributions over bit-strings $D^0 = \{D^0_\eta\}_{\eta\in\mathbb{N}}$ and $D^1 = \{D^1_\eta\}_{\eta\in\mathbb{N}}$ are computationally indistinguishable if for all PPT algorithms $\mathcal{A}$:

$$\Pr[b = b^* \mid b \in_R \{0,1\}, x \leftarrow D^b_\eta, b^* \leftarrow \mathcal{A}(1^\eta, x)] = 1/2 + \varepsilon(\eta)$$

for some negligible function ε.
A function ε is negligible if

$$\lim_{\eta\to\infty} \varepsilon(\eta)\cdot p(\eta) = 0$$

for all polynomials p.

Decomposing a formal expression

$$e_1 \vdash e_2$$

The value of e_1 tells us the value of e_2

Decomposing a formal expression

$$e_1 \vdash e_2$$

The value of e_1 tells us the value of e_2

$$\begin{aligned} e &\vdash e \\ e \vdash (e_1, e_2) &\Rightarrow e \vdash e_1 \;\wedge\; e \vdash e_2 \\ e \vdash \{e'\}_k^r \;\wedge\; e \vdash k &\Rightarrow e \vdash e' \end{aligned}$$

Decomposing a formal expression

$$e_1 \vdash e_2$$

The value of e_1 tells us the value of e_2

$$\begin{aligned} e &\vdash e \\ e \vdash (e_1, e_2) &\Rightarrow e \vdash e_1 \ \wedge \ e \vdash e_2 \\ e \vdash \{e'\}_k^r \ \wedge \ e \vdash k &\Rightarrow e \vdash e' \end{aligned}$$

Examples:

$$(\{\mathsf{1011}\}_{k_1}^r, \{k_1\}_{k_2}^{r'}, k_2) \vdash \mathsf{1011}$$

$$(\{\mathsf{1011}\}_{k_1}^r, \{k_1\}_{k_2}^{r'}, \{k_2\}_{k_3}^{r''}) \not\vdash \mathsf{1011}$$

$$(\{\mathsf{1011}\}_{k_1}^r, \{k_1\}_{k_2}^{r'}, \{k_2\}_{k_1}^{r''}) \not\vdash \mathsf{1011}$$

Decomposing a formal expression

$$e_1 \vdash e_2$$

The value of e_1 tells us the value of e_2

$$\begin{aligned} e &\vdash e \\ e \vdash (e_1, e_2) &\Rightarrow e \vdash e_1 \ \wedge \ e \vdash e_2 \\ e \vdash \{e'\}_k^r \ \wedge \ e \vdash k &\Rightarrow e \vdash e' \end{aligned}$$

Examples:

$$(\{\mathsf{1011}\}_{k_1}^r, \{k_1\}_{k_2}^{r'}, k_2) \vdash \mathsf{1011}$$
$$(\{\mathsf{1011}\}_{k_1}^r, \{k_1\}_{k_2}^{r'}, \{k_2\}_{k_3}^{r''}) \not\vdash \mathsf{1011}$$
$$(\{\mathsf{1011}\}_{k_1}^r, \{k_1\}_{k_2}^{r'}, \{k_2\}_{k_1}^{r''}) \not\vdash \mathsf{1011}$$

Let $openkeys(e) = \{k \in \mathbf{Keys} \,|\, e \vdash k\}$.

The pattern of a formal expression

- Enlarge the set $\mathbf{Exp}$: $\quad e ::= \ldots \mid \square^r$.
- For a set $K \subseteq \mathbf{Keys}$ define

$$
\begin{aligned}
pat(k, K) &= k \\
pat(b, K) &= b \\
pat((e_1, e_2), K) &= (pat(e_1, K), pat(e_2, K)) \\
pat(\{e\}_k^r, K) &= \begin{cases} \{pat(e, K)\}_k^r, & \text{if } k \in K \\ \square^r, & \text{if } k \notin K \end{cases}
\end{aligned}
$$

- Let $pattern(e) = pat(e, openkeys(e))$.

The pattern of a formal expression

- Enlarge the set $\mathbf{Exp}$: $\quad e ::= \ldots | \square^r$.
- For a set $K \subseteq \mathbf{Keys}$ define

$$
\begin{aligned}
pat(k, K) &= k \\
pat(b, K) &= b \\
pat((e_1, e_2), K) &= (pat(e_1, K), pat(e_2, K)) \\
pat(\{e\}_k^r, K) &= \begin{cases} \{pat(e, K)\}_k^r, & \text{if } k \in K \\ \square^r, & \text{if } k \notin K \end{cases}
\end{aligned}
$$

- Let $pattern(e) = pat(e, openkeys(e))$.
- Define $e_1 \cong e_2$ if $pattern(e_1) = pattern(e_2)\sigma_K\sigma_R$ for some
  - σ_K — a permutation of the keys $\mathbf{Keys}$;
  - σ_R — a permutation of the random coins $\mathbf{Coins}$.

Examples

$$pattern((\{\texttt{1011}\}^{r}_{k_1}, \{k_1\}^{r'}_{k_2}, k_2)) = (\{\texttt{1011}\}^{r}_{k_1}, \{k_1\}^{r'}_{k_2}, k_2)$$
$$pattern((\{\texttt{1011}\}^{r}_{k_1}, \{k_1\}^{r'}_{k_2}, \{k_2\}^{r''}_{k_3})) = (\Box^{r}, \Box^{r'}, \Box^{r''})$$
$$pattern((\{\texttt{1011}\}^{r}_{k_1}, \{k_1\}^{r'}_{k_2}, \{k_2\}^{r''}_{k_1})) = (\Box^{r}, \Box^{r'}, \Box^{r''})$$
$$pattern((\{\texttt{1}\}^{r_1}_{k_2}, \{k_2\}^{r_2}_{k_3}, \{\{\texttt{0}\}^{r_4}_{k_2}\}^{r_3}_{k_1}, k_1)) = (\Box^{r_1}, \Box^{r_2}, \{\Box^{r_4}\}^{r_3}_{k_1}, k_1)$$
$$pattern((\{k_4, \texttt{0}\}^{r_1}_{k_3}, \{k_3\}^{r_2}_{k_2}, \{\{\texttt{11}\}^{r_4}_{k_4}\}^{r_3}_{k_1}, k_1)) = (\Box^{r_1}, \Box^{r_2}, \{\Box^{r_4}\}^{r_3}_{k_1}, k_1)$$

IND-CPA-security of an encryption scheme

- Encrypting oracle $\mathcal{O}_1^{\mathrm{IND-CPA}}$:
  Initialization: $\mathtt{k} \leftarrow \mathcal{K}(1^\eta)$
  method encrypt($\mathtt{x}$)
  $\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}, \mathtt{x})$
  return $\mathtt{y}$
- Constant-encrypting oracle $\mathcal{O}_0^{\mathrm{IND-CPA}}$:
  Initialization: $\mathtt{k} \leftarrow \mathcal{K}(1^\eta)$
  method encrypt($\mathtt{x}$)
  $l := \mathit{length}(\mathtt{x})$
  $\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}, \mathtt{0}^l)$
  return $\mathtt{y}$

$(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is IND-CPA-secure if for all PPT algorithms $\mathcal{A}$ exists a negligible ε, such that

$$\Pr[b = b^* \mid b \in_R \{0,1\}, b^* \leftarrow \mathcal{A}^{\mathcal{O}_b^{\mathrm{IND-CPA}}}(1^\eta)] = 1/2 + \varepsilon(\eta)$$

IND-CPA-security of an encryption scheme

- Encrypting oracle $\mathcal{O}_1^{\mathrm{IND-CPA}}$:
  Initialization: $\mathtt{k} \leftarrow \mathcal{K}(1^\eta)$
  method encrypt$(\mathtt{x})$
  $\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}, \mathtt{x})$
  return $\mathtt{y}$
- Constant-encrypting oracle $\mathcal{O}_0^{\mathrm{IND-CPA}}$:
  Initialization: $\mathtt{k} \leftarrow \mathcal{K}(1^\eta)$
  method encrypt$(\mathtt{x})$
  $l := \mathit{length}(\mathtt{x})$
  $\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}, \mathtt{0}^l)$
  return $\mathtt{y}$

$(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is IND-CPA-secure if for all PPT algorithms $\mathcal{A}$ exists a negligible ε, such that

$$\Pr[b = b^* \mid b \in_R \{0,1\}, b^* \leftarrow \mathcal{A}^{\mathcal{O}_b^{\mathrm{IND-CPA}}}(1^\eta)] = 1/2 + \varepsilon(\eta)$$

In other words: $\mathcal{O}_1^{\mathrm{IND-CPA}} \approx \mathcal{O}_0^{\mathrm{IND-CPA}}$.

Hiding the identities of keys

- Oracle with two keys $\mathcal{O}_1^{\mathrm{hide-key}}$:

Initialization:
$\mathtt{k}_1 \leftarrow \mathcal{K}(1^\eta)$
$\mathtt{k}_2 \leftarrow \mathcal{K}(1^\eta)$

method encrypt1$(\mathtt{x})$
$\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}_1, \mathtt{x})$
return $\mathtt{y}$

method encrypt2$(\mathtt{x})$
$\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}_2, \mathtt{x})$
return $\mathtt{y}$

- Oracle with one key $\mathcal{O}_0^{\mathrm{hide-key}}$:

Initialization:
$\mathtt{k} \leftarrow \mathcal{K}(1^\eta)$

method encrypt1$(\mathtt{x})$
$\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}, \mathtt{x})$
return $\mathtt{y}$

method encrypt2$(\mathtt{x})$
$\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}, \mathtt{x})$
return $\mathtt{y}$

$(\mathcal{K}, \mathcal{E}, \mathcal{D})$ hides the identities of keys / is which-key concealing if $\mathcal{O}_1^{\mathrm{hide-key}} \approx \mathcal{O}_0^{\mathrm{hide-key}}$.

Hiding the identities of keys

- Oracle with two keys $\mathcal{O}_1^{\mathrm{hide-key}}$:

Initialization:	**method** encrypt1$(\mathtt{x})$	**method** encrypt2$(\mathtt{x})$
$\mathtt{k}_1 \leftarrow \mathcal{K}(1^\eta)$	$\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}_1, \mathtt{x})$	$\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}_2, \mathtt{x})$
$\mathtt{k}_2 \leftarrow \mathcal{K}(1^\eta)$	**return** $\mathtt{y}$	**return** $\mathtt{y}$

- Oracle with one key $\mathcal{O}_0^{\mathrm{hide-key}}$:

Initialization:	**method** encrypt1$(\mathtt{x})$	**method** encrypt2$(\mathtt{x})$
$\mathtt{k} \leftarrow \mathcal{K}(1^\eta)$	$\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}, \mathtt{x})$	$\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}, \mathtt{x})$
	return $\mathtt{y}$	**return** $\mathtt{y}$

$(\mathcal{K}, \mathcal{E}, \mathcal{D})$ hides the identities of keys / is which-key concealing if $\mathcal{O}_1^{\mathrm{hide-key}} \approx \mathcal{O}_0^{\mathrm{hide-key}}$.

IND-CPA-secure which-key concealing encryption schemes are easily constructed (CCA- or CTR-mode of operation of block ciphers).

Hiding the length of the plaintext

- An encryption scheme is length-concealing if the length of the plaintext cannot be determined from the ciphertext.
- Achievable by padding the plaintexts.
  - Questionable for nested encryptions. . .
- For simplicity, we will assume that our encryption scheme is length-concealing.
  - And also which-key concealing and IND-CPA-secure.
- Otherwise we'd need to define lengths of formal expressions.
  - Not difficult, but currently not so interesting

IND-CPA, which-key and length-concealing:

Let $\mathbf{0}$ be a fixed bit-string.

- Oracle $\mathcal{O}_1^{\mathrm{type-0}}$:

  Initialization:
  $\mathtt{k}_1 \leftarrow \mathcal{K}(1^\eta)$
  $\mathtt{k}_2 \leftarrow \mathcal{K}(1^\eta)$

  method encrypt1$(\mathtt{x})$
  $\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}_1, \mathtt{x})$
  return $\mathtt{y}$

  method encrypt2$(\mathtt{x})$
  $\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}_2, \mathtt{x})$
  return $\mathtt{y}$

- Oracle $\mathcal{O}_0^{\mathrm{type-0}}$:

  Initialization:
  $\mathtt{k} \leftarrow \mathcal{K}(1^\eta)$

  method encrypt1$(\mathtt{x})$
  $\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}, \mathbf{0})$
  return $\mathtt{y}$

  method encrypt2$(\mathtt{x})$
  $\mathtt{y} \leftarrow \mathcal{E}(\mathtt{k}, \mathbf{0})$
  return $\mathtt{y}$

$(\mathcal{K}, \mathcal{E}, \mathcal{D})$ has all three listed properties if $\mathcal{O}_1^{\mathrm{type-0}} \approx \mathcal{O}_0^{\mathrm{type-0}}$.

Semantics of expressions and patterns

- For each $k \in \mathbf{Keys}$ let $\mathtt{s}_k \leftarrow \mathcal{K}(1^\eta)$
- For each $r \in \mathbf{Coins}$ let $\mathtt{s}_r \in_R \{0,1\}^\omega$
- Let $\mathtt{k}_\Box \leftarrow \mathcal{K}(1^\eta)$.

Define

$$
\begin{aligned}
[\![k]\!]_\eta &= \mathtt{s}_k \\
[\![b]\!]_\eta &= b \\
[\![(e_1, e_2)]\!]_\eta &= \langle [\![e_1]\!]_\eta, [\![e_2]\!]_\eta \rangle \\
[\![\{e'\}_k^r]\!]_\eta &= \mathcal{E}^{\mathtt{s}_r}(\eta, \mathtt{s}_k, [\![e']\!]_\eta) \\
\hline
[\![\Box^r]\!]_\eta &= \mathcal{E}^{\mathtt{s}_r}(\eta, \mathtt{k}_\Box, \mathbf{0})
\end{aligned}
$$

Theorem of equivalence

Theorem. Let $e_1, e_2 \in \mathbf{Exp}$. If $e_1 \cong e_2$ then* $[\![e_1]\!] \approx [\![e_2]\!]$.

Replacing one key

- For a key $\overline{k} \in \mathbf{Keys}$ define

$$
\begin{aligned}
replacekey(k, \overline{k}) &= k \\
replacekey(b, \overline{k}) &= b \\
replacekey((e_1, e_2), \overline{k}) &= (replacekey(e_1, \overline{k}), replacekey(e_2, \overline{k})) \\
replacekey(\{e\}_k^r, \overline{k}) &= \begin{cases} \Box^r, & \text{if } k = \overline{k} \\ \{replacekey(e, \overline{k})\}_k^r, & \text{if } k \neq \overline{k} \end{cases} \\
replacekey(\Box^r, \overline{k}) &= \Box^r
\end{aligned}
$$

- **Lemma.** Let $e \in \mathbf{Exp}$. Let key $\overline{k}$ occur in e only as encryption key. Then $[\![e]\!] \approx [\![replacekey(e, \overline{k})]\!]$.

Proof of the lemma

Assume that $\mathcal{B}$ distinguishes $[\![e]\!]$ from $[\![\mathit{replacekey}(e,\overline{k})]\!]$.
Let $\mathcal{A}^{\mathcal{O}}(\eta)$ work as follows:

- Initialize:
  - Let $\mathtt{s}_k \leftarrow \mathcal{K}(\eta)$ for all keys k occurring in e, except $\overline{k}$.
  - Let $\mathtt{s}_r \in_R \{0,1\}^\omega$ for all r occurring in e, except as $\{\ldots\}^r_{\overline{k}}$.
  - Let $\mathtt{k}_\square \leftarrow \mathcal{K}(1^\eta)$.
- Let $L = \{\}$ (empty mapping).
- Compute the "semantics" v of e as follows by invoking $\textsc{Sem}^{\mathcal{O}}(e)$
  - $\textsc{Sem}^{\mathcal{O}}(e) = [\![e]\!]$ if $\mathcal{O} = \mathcal{O}_1^{\mathrm{type-0}}$.
  - $\textsc{Sem}^{\mathcal{O}}(e) = [\![\mathit{replacekey}(e,\overline{k})]\!]$ if $\mathcal{O} = \mathcal{O}_0^{\mathrm{type-0}}$.
- **return** $\mathcal{B}(\eta, v)$.

$\mathcal{A}$ can distinguish $\mathcal{O}_1^{\mathrm{type-0}}$ and $\mathcal{O}_0^{\mathrm{type-0}}$ as well as $\mathcal{B}$ can distinguish $[\![e]\!]$ and $[\![\mathit{replacekey}(e,\overline{k})]\!]$.

Computing $[\![e]\!]$ or $[\![replacekey(e,\overline{k})]\!]$

$\textsc{Sem}^{\mathcal{O}}(e)$ is: **case** e **of**

- k: **return** $\mathtt{s}_k$ *(note that $k \neq \overline{k}$)*
- b: **return** b
- (e_1, e_2): let $v_i = \textsc{Sem}^{\mathcal{O}}(e_i)$; **return** $\langle v_1, v_2 \rangle$
- $\{e\}_k^r$: let $v = \textsc{Sem}^{\mathcal{O}}(e)$;
  - If $k \neq \overline{k}$ then **return** $\mathcal{E}^{\mathtt{s}_r}(\eta, \mathtt{s}_k, v)$
  - If $k = \overline{k}$ and $L(r)$ is not defined then
    - let $L(r) = \mathcal{O}.\mathrm{encrypt1}(v)$;
    - **return** $L(r)$
  - If $k = \overline{k}$ and $L(r)$ is defined then **return** $L(r)$
- $\Box^r$: **return** $\mathcal{O}.\mathrm{encrypt2}(\mathbf{0})$

Proof of the theorem

1. $replacekey(replacekey(\cdots replacekey(e, k_1), k_2) \cdots, k_n) = pattern(e)$
   if $\{k_1, \ldots, k_n\}$ are all keys in e that the adversary cannot obtain.
   Denote this set of keys by $hidkeys(e)$.
2. Apply the **lemma** sequentially to each key in $hidkeys(e)$, thereby establishing

$$\boxed{[\![e]\!] \approx [\![pattern(e)]\!].}$$

   * In general, not all orders of keys in $hidkeys(e)$ are suitable.

3. Permuting the formal keys and coins does not change the generated probability distribution over bit-strings.

If $e_1 \cong e_2$ then* $[\![e_1]\!] \approx [\![pattern(e_1)]\!] = [\![pattern(e_2)]\!] = [\![e_2]\!]$.

Example 1

$$[\![(\{k_4, \mathtt{0}\}_{k_3}^{r_1}, \{k_3\}_{k_2}^{r_2}, \{\{\mathtt{1}\,\mathtt{1}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$$[\![(\{\mathtt{1}\}_{k_2}^{r_1}, \{k_2\}_{k_3}^{r_2}, \{\{\mathtt{0}\}_{k_2}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

Example 1

$$[\![(\{k_4, \mathtt{0}\}_{k_3}^{r_1}, \{k_3\}_{\color{red}{k_2}}^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$$[\![(\{\mathtt{1}\}_{k_2}^{r_1}, \{k_2\}_{k_3}^{r_2}, \{\{\mathtt{0}\}_{k_2}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

Example 1

$$[\![(\{k_4, \mathtt{0}\}_{k_3}^{r_1}, \{k_3\}_{k_2}^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$$\approx$$

$$[\![(\{k_4, \mathtt{0}\}_{k_3}^{r_1}, \square^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$$[\![(\{\mathtt{1}\}_{k_2}^{r_1}, \{k_2\}_{k_3}^{r_2}, \{\{\mathtt{0}\}_{k_2}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

Example 1

$$[\![(\{k_4, \mathtt{0}\}_{k_3}^{r_1}, \{k_3\}_{k_2}^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$$\approx$$

$$[\![(\{k_4, \mathtt{0}\}_{\color{red}{k_3}}^{r_1}, \Box^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$$[\![(\{\mathtt{1}\}_{k_2}^{r_1}, \{k_2\}_{k_3}^{r_2}, \{\{\mathtt{0}\}_{k_2}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

Example 1

$$[\![(\{k_4, \mathtt{0}\}_{k_3}^{r_1}, \{k_3\}_{k_2}^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$$\approx$$

$$[\![(\{k_4, \mathtt{0}\}_{k_3}^{r_1}, \Box^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$$\approx$$

$$[\![(\Box^{r_1}, \Box^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$$[\![(\{\mathtt{1}\}_{k_2}^{r_1}, \{k_2\}_{k_3}^{r_2}, \{\{\mathtt{0}\}_{k_2}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

Example 1

$$\llbracket(\{k_4, \mathtt{0}\}_{k_3}^{r_1}, \{k_3\}_{k_2}^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)\rrbracket$$

$$\approx$$

$$\llbracket(\{k_4, \mathtt{0}\}_{k_3}^{r_1}, \Box^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)\rrbracket$$

$$\approx$$

$$\llbracket(\Box^{r_1}, \Box^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)\rrbracket$$

$$\approx$$

$$\llbracket(\Box^{r_1}, \Box^{r_2}, \{\Box^{r_4}\}_{k_1}^{r_3}, k_1)\rrbracket$$

$$\llbracket(\{\mathtt{1}\}_{k_2}^{r_1}, \{k_2\}_{k_3}^{r_2}, \{\{\mathtt{0}\}_{k_2}^{r_4}\}_{k_1}^{r_3}, k_1)\rrbracket$$

Example 1

$$\llbracket(\{k_4, \mathtt{0}\}_{k_3}^{r_1}, \{k_3\}_{k_2}^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)\rrbracket$$

$$\approx$$

$$\llbracket(\{k_4, \mathtt{0}\}_{k_3}^{r_1}, \Box^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)\rrbracket$$

$$\approx$$

$$\llbracket(\Box^{r_1}, \Box^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)\rrbracket$$

$$\approx$$

$$\llbracket(\Box^{r_1}, \Box^{r_2}, \{\Box^{r_4}\}_{k_1}^{r_3}, k_1)\rrbracket$$

$$\llbracket(\{\mathtt{1}\}_{k_2}^{r_1}, \{k_2\}_{k_3}^{r_2}, \{\{\mathtt{0}\}_{k_2}^{r_4}\}_{k_1}^{r_3}, k_1)\rrbracket$$

Example 1

$$[\![(\{k_4, \mathtt{0}\}_{k_3}^{r_1}, \{k_3\}_{k_2}^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$$\approx$$

$$[\![(\{k_4, \mathtt{0}\}_{k_3}^{r_1}, \square^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$$\approx$$

$$[\![(\square^{r_1}, \square^{r_2}, \{\{\mathtt{11}\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$$\approx$$

$$[\![(\square^{r_1}, \square^{r_2}, \{\square^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$$\approx$$

$$[\![(\{\mathtt{1}\}_{k_2}^{r_1}, \square^{r_2}, \{\{\mathtt{0}\}_{k_2}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$$\approx$$

$$[\![(\{\mathtt{1}\}_{k_2}^{r_1}, \{k_2\}_{k_3}^{r_2}, \{\{\mathtt{0}\}_{k_2}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

Example 2

$$pattern((\{k_3\}_{k_2}^{r_1}, \{k_4\}_{k_3}^{r_2}, \{\{k_2\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)) = (\square^{r_1}, \square^{r_2}, \{\square^{r_4}\}_{k_1}^{r_3}, k_1)$$

Example 2

$$pattern((\{k_3\}_{k_2}^{r_1}, \{k_4\}_{k_3}^{r_2}, \{\{k_2\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)) = (\Box^{r_1}, \Box^{r_2}, \{\Box^{r_4}\}_{k_1}^{r_3}, k_1)$$

$$[\![(\{k_3\}_{k_2}^{r_1}, \{k_4\}_{k_3}^{r_2}, \{\{k_2\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

Example 2

$$pattern((\{k_3\}_{k_2}^{r_1}, \{k_4\}_{k_3}^{r_2}, \{\{k_2\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)) = (\square^{r_1}, \square^{r_2}, \{\square^{r_4}\}_{k_1}^{r_3}, k_1)$$

$$[\![(\{k_3\}_{k_2}^{r_1}, \{k_4\}_{k_3}^{r_2}, \{\{k_2\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)]\!]$$

$\langle$cannot apply the lemma$\rangle$

Encryption cycles

- Let e be a formal expression.
- Consider the following directed graph $G = (V, E)$:
  - $V = hidkeys(e)$
  - $(k_i \longrightarrow k_j) \in E$ if e has a subexpression of the form

$$\{\cdots k_j \cdots\}_{k_i}^{r}$$

    (we say that k_i encrypts k_j)

- e has no encryption cycles if G does not contain directed cycles.

Encryption cycles

- Let e be a formal expression.
- Consider the following directed graph $G = (V, E)$:
  - $V = hidkeys(e)$
  - $(k_i \longrightarrow k_j) \in E$ if e has a subexpression of the form

$$\{\cdots k_j \cdots\}^r_{k_i}$$

    (we say that k_i encrypts k_j)
- e has no encryption cycles if G does not contain directed cycles.

Theorem. If e contains no encryption cycles then $[\![e]\!] \approx [\![pattern(e)]\!]$.

Encryption cycles

- Let e be a formal expression.
- Consider the following directed graph $G = (V, E)$:
  - $V = hidkeys(e)$
  - $(k_i \rightarrow k_j) \in E$ if e has a subexpression of the form

$$\{\cdots k_j \cdots\}_{k_i}^{r}$$

    (we say that k_i encrypts k_j)

- e has no encryption cycles if G does not contain directed cycles.

Theorem. If e contains no encryption cycles then $[\![e]\!] \approx [\![pattern(e)]\!]$.

"No encryption cycles" is sufficient, but not necessary condition for the sequential applicability of our lemma.

Example: $(\{k_3\}_{k_2}^{r_1}, \{k_4\}_{k_3}^{r_2}, \{\{k_2\}_{k_4}^{r_4}\}_{k_1}^{r_3})$ is OK.

Severity of encryption cycles

Exercise. Take an encryption scheme that is assumed to be IND-CPA-secure. Modify it so, that it is still IND-CPA-secure, but defenseless against an adversary that has somehow obtained $\{k\}_k$.

Dealing with encryption cycles

- We could increase the relation $\vdash$
  - Thereby allowing the adversary to “break encryption cycles”.
- We could strengthen the security definition of the symmetric encryption scheme
  - KDM-IND-CPA-security
  - key-dependent messages
  - Is such definition instantiable?

Breaking encryption cycles

Define the relations $\vdash_{\mathbf{K}}$ for any set $\mathbf{K}$ of formal keys as follows:

$$
\begin{aligned}
e &\vdash_{\mathbf{K}} e \\
e \vdash_{\mathbf{K}} (e_1, e_2) &\Rightarrow e \vdash_{\mathbf{K}} e_1 \wedge e \vdash_{\mathbf{K}} e_2 \\
e \vdash_{\mathbf{K}} e' &\Rightarrow e \vdash_{\mathbf{K} \cup \mathbf{K}'} e' \\
e \vdash_{\mathbf{K}} \{e'\}_k^r &\Rightarrow e \vdash_{\mathbf{K} \cup \{k\}} e' \\
e \vdash_{\mathbf{K} \cup \{k\}} e' \wedge e \vdash_{\mathbf{K}} k &\Rightarrow e \vdash_{\mathbf{K}} e' \\
e \vdash_{\mathbf{K} \cup \{k\}} k &\Rightarrow e \vdash_{\mathbf{K}} k
\end{aligned}
$$

And define $\vdash$ as the relation $\vdash_{\emptyset}$.

Exercise. What is the pattern of messages $(\{k_3\}_{k_2}^{r_1}, \{k_4\}_{k_3}^{r_2}, \{\{k_2\}_{k_4}^{r_4}\}_{k_1}^{r_3}, k_1)$ and $(\{k_3\}_{k_2}^{r_1}, \{k_4\}_{k_3}^{r_2}, \{\{k_2\}_{k_4}^{r_4}\}_{k_1}^{r_3})$ by the new definition of $\vdash$?

KDM-IND-CPA-security

- Defined as the indistinguishability of certain two encrypting oracles $\mathcal{O}_0$ and $\mathcal{O}_1$.
- Both "initially create" an array $\mathbf{k}[0..\infty]$ of fresh keys.
- A query to an oracle is a pair (j, g), where
  - $j \in \mathbb{N}$
  - g is a program that returns a bit-string
    - g may refer to $\mathbf{k}$.
    - the length of g's output may not depend on $\mathbf{k}$.
- $\mathcal{O}_1$ returns $\mathcal{E}_{\mathbf{k}[j]}(g(\mathbf{k}))$ to the query (j, g).
- $\mathcal{O}_0$ returns $\mathcal{E}_{\mathbf{k}[j]}(0^{|g(\mathbf{k})|})$ to the query (j, g).

(this definition allows $\mathcal{E}$ to reveal the lengths of plaintexts and identities of keys)

Achieving KDM-IND-CPA-security

- Simple in the random oracle model
  - Let $H(x)$ denote random oracle's output for the query x
  - The program g may also contain instructions to call H
- Let $\mathcal{K}(\eta)$ just output a random element of $\{0,1\}^\eta$.
- Let $\mathcal{E}^r(\eta, k, x) = (r, H(k\|r) \oplus x)$
  - Assume that the output of H has the same length as x
  - **Exercise.** How do we construct such a H from some random oracle H_0 whose output length is fixed?

Exercise. Show that this scheme is KDM-IND-CPA-secure.

- It is not known how to achieve KDM-security in the plain model.
- Possible, if we restrict the shape of g in a certain way.
- This restricted set can still be large enough to contain the computation of $[\![\cdot]\!]$.

Table of Contents

- The Abadi-Rogaway result on the indistinguishability of computational interpretations of formal messages.
- **Translating protocol traces between formal and computational world.**

Public-key primitives

- Extend the construction of the set of formal messages by
  - keypairs $kp \in \mathbf{EKeys}$ for encryption and $kp \in \mathbf{SKeys}$ for signing;
  - operations kp^+ and kp^- to take the public and secret components of keys;
  - public-key encryptions $\{\!|e|\!\}^r_{kp^+}$ and signatures $[\!\{e\}\!]^r_{kp^-}$.
- Fix a public-key encryption scheme $(\mathcal{K}_\mathrm{p}, \mathcal{E}_\mathrm{p}, \mathcal{D}_\mathrm{p})$ and a signature scheme $(\mathcal{K}_\mathrm{s}, \mathcal{S}_\mathrm{s}, \mathcal{V}_\mathrm{s})$.
  - Use $\mathcal{K}_\mathrm{p}$, $\mathcal{E}_\mathrm{p}$, $\mathcal{K}_\mathrm{s}$, $\mathcal{K}_\mathrm{s}$ to define the semantics of new constructs.
- Similar results can be obtained with $\{\!|\cdot|\!\}_\cdot$ in messages.
  - If secret keys are not part of messages then encryption cycles are not an issue.

Specifying the protocols

- A set $\mathcal{P}$ of principals (some of them possibly corrupted). Each one with fixed keypairs for signing and encryption.
  - There are keys $\mathsf{ek}(P)$, $\mathsf{dk}(P)$, $\mathsf{sk}(P)$, $\mathsf{vk}(P)$ for each principal P.
- A set of roles.
  - A list of pairs of incoming and outgoing messages.
  - May contain nonces.
  - Also may contain message variables and principal variables.

Example roles

Needham-Schroeder-Lowe public-key protocol:

$$
\begin{aligned}
A \longrightarrow B &: \{[N_A, A]\}_{\mathsf{ek}(B)}\\
B \longrightarrow A &: \{[N_A, N_B, B]\}_{\mathsf{ek}(A)}\\
A \longrightarrow B &: \{[N_B]\}_{\mathsf{ek}(B)}
\end{aligned}
$$

- Initiator role:

$$
(\mathit{Start}, \{[N_A, X_{\mathrm{Init}}]\}_{\mathsf{ek}(X_{\mathrm{Resp}})})
$$
$$
(\{[N_A, X_N, X_{\mathrm{Resp}}]\}_{\mathsf{ek}(X_{\mathrm{Init}})}, \{[X_N]\}_{\mathsf{ek}(X_{\mathrm{Resp}})})
$$

- Responder role:

$$
(\{[X_N, X_{\mathrm{Init}}]\}_{\mathsf{ek}(X_{\mathrm{Resp}})}, \{[X_N, N_B, X_{\mathrm{Resp}}]\}_{\mathsf{ek}(X_{\mathrm{Init}})})
$$
$$
(\{[N_B]\}_{\mathsf{ek}(X_{\mathrm{Resp}})}, \mathit{Ok})
$$

Execution

- Adversary may start new runs by stating $\mathbf{new}(sid; P_1, \ldots, P_n)$.
  - sid is the unique session identifier of the run.
  - $P_1, \ldots, P_n$ are names of principals that fulfill the roles $R_1, \ldots, R_n$.

Execution

- Adversary may start new runs by stating $\mathbf{new}(sid; P_1, \ldots, P_n)$.
  - sid is the unique session identifier of the run.
  - $P_1, \ldots, P_n$ are names of principals that fulfill the roles $R_1, \ldots, R_n$.
- Adversary may send messages by stating $\mathbf{recv}(sid, R_i, m)$ where m is a message.
  - The role R_i in the run sid will receive the message m and process it.

Execution

- Adversary may start new runs by stating $\mathbf{new}(sid; P_1, \ldots, P_n)$.
  - sid is the unique session identifier of the run.
  - $P_1, \ldots, P_n$ are names of principals that fulfill the roles $R_1, \ldots, R_n$.
- Adversary may send messages by stating $\mathbf{recv}(sid, R_i, m)$ where m is a message.
  - The role R_i in the run sid will receive the message m and process it.
- When a principal P_i running the role $R_i = (m_{\mathrm{i}}, m_{\mathrm{o}}) :: R_i'$ in the run sid will receive a message m, then it will
  - match m with m_{i};
  - generate a new message m' by instantiating the outgoing message m_{o} and send it: $\mathbf{send}(sid, R_i, m')$;
  - Set R_i to R_i' (in sid only).

Execution

- Decompose m according to m_{i}.
  - Use $\mathrm{dk}(P_i)$ to decrypt messages encrypted with $\mathrm{ek}(P_i)$.
  - The keys for symmetric encryption are contained in m_{i}.
- Verify the equality of instantiated parts of m_{i} to the corresponding parts of m'.
- Initialize the new variables in m_{i} with the corresponding parts of m'.
- Verify the signatures in m'.

re m

- When a principal P_i running the role $R_i = (m_{\mathrm{i}}, m_{\mathrm{o}}) :: R'_i$ in the run sid will receive a message m, then it will
  - match m with m_{i};
  - generate a new message m' by instantiating the outgoing message m_{o} and send it: $\mathbf{send}(sid, R_i, m')$;
  - Set R_i to R'_i (in sid only).

Execution

- Adversary may start new runs by stating $\mathbf{new}(sid; P_1, \ldots, P_n)$.
  - sid is the unique session identifier of the run.
  - $P_1, \ldots, P_n$ are names of principals that fulfill the roles $R_1, \ldots, R_n$.
- Adversary may send messages by stating $\mathbf{recv}(sid, R_i, m)$ where m is a message.
  - The role R_i in the run sid will receive the message m and process it.
- When a principal P_i running the role $R_i = (m_\mathrm{i}, m_\mathrm{o}) :: R_i'$ in the run sid will receive a message m, then it will
  - match m with m_i;
  - generate a new message m' by instantiating the outgoing message m_o and send it: $\mathbf{send}(sid, R_i, m')$;
  - Set R_i to R_i' (in sid only).

Execution

- Adversary may start new runs by stating $\mathbf{new}(sid; P_1, \ldots, P_n)$.
  - sid is the unique session identifier of the run.
  - $P_1, \ldots, P_n$ are names of principals that fulfill the roles $R_1, \ldots, R_n$.

re m

- Use the values of already known keys, nonces, variables, etc.
- Generate new values for keys and nonces that occur first time in m_o.

- When a principal P_i running the role $R_i = (m_\mathrm{i}, m_\mathrm{o}) :: R_i'$ in the run sid will receive a message m, then it will
  - match m with m_i;
  - generate a new message m' by instantiating the outgoing message m_o and send it: $\mathbf{send}(sid, R_i, m')$;
  - Set R_i to R_i' (in sid only).

Execution traces

- An execution trace is a sequence of **new**-, **recv**- and **send**-statements.
- We have traces in both models — there are
  - formal traces — sequences of terms over a message algebra with a countable number of atoms for keys, nonces, random coins;
  - computational traces — sequences of bit-strings.
- A formal trace is valid if each message in a **recv**-statement can be generated from messages in previous **send**- and **recv**-statements.

Translating Formal $\rightarrow$ Computational

- A formal trace t^f is a sequence consisting of principal names and formal messages.
- Formal messages are made up of formal nonces, formal keys, formal encryptions and decryptions using formal coins.
- Fix a mapping c from formal constants, nonces, keys and coins to bit-strings.
- Extend c to the entire trace, giving the computational trace $c(t^f)$.
- Denote $t^f \leq t^c$ if the computational trace t^c can be obtained as a translation of the formal trace t^f.

Translating Formal $\rightarrow$ Computational

- A formal trace t^f is a sequence consisting of principal names and formal messages.
- Formal messages are made up of formal nonces, formal keys, formal encryptions and decryptions using formal coins.
- Fix a mapping c from formal constants, nonces, keys and coins to bit-strings.
- Extend c to the entire trace, giving the computational trace $c(t^f)$.
- Denote $t^f \leq t^c$ if the computational trace t^c can be obtained as a translation of the formal trace t^f.

Lemma. If the used cryptographic primitives are secure then for any computational adversary $\mathcal{A}$, if t^c is a computational trace of the protocol running together with $\mathcal{A}$ then with overwhelming probability there exists a valid formal trace t^f, such that $t^f \leq t^c$.

Security of primitives

- The encryption systems must be IND-CCA secure.
    - Adversary may not be able to distinguish $\mathcal{E}(k, \pi_1(\cdot, \cdot))$ and $\mathcal{E}(k, \pi_2(\cdot, \cdot))$ even with access to $\mathcal{D}(k, \cdot)$.
    - Results from the encryption oracle may not be submitted to the decryption oracle.

Security of primitives

- The encryption systems must be IND-CCA secure.
  - Adversary may not be able to distinguish $\mathcal{E}(k, \pi_1(\cdot,\cdot))$ and $\mathcal{E}(k, \pi_2(\cdot,\cdot))$ even with access to $\mathcal{D}(k, \cdot)$.
  - Results from the encryption oracle may not be submitted to the decryption oracle.
- The signature system must be EF-CMA secure.
  - Adversary may not be able to produce a valid (message,signature)-pair, even when interacting with a signing oracle.
  - Messages submitted to the oracle do not count.

Security of primitives

- The encryption systems must be IND-CCA secure.
  - Adversary may not be able to distinguish $\mathcal{E}(k, \pi_1(\cdot,\cdot))$ and $\mathcal{E}(k, \pi_2(\cdot,\cdot))$ even with access to $\mathcal{D}(k, \cdot)$.
  - Results from the encryption oracle may not be submitted to the decryption oracle.
- The signature system must be EF-CMA secure.
  - Adversary may not be able to produce a valid (message,signature)-pair, even when interacting with a signing oracle.
  - Messages submitted to the oracle do not count.
- The message must be recoverable from the signature (and the verification key).

Translating Computational $\rightarrow$ Formal

Consider

- a computational trace,
  - Actually, the set $\mathcal{M}$ of messages appearing in it.
- the set $\mathcal{K}$ of secret decryption keys of participants.

Iterate:

Translating Computational $\rightarrow$ Formal

Consider

- a computational trace,
  - Actually, the set $\mathcal{M}$ of messages appearing in it.
- the set $\mathcal{K}$ of secret decryption keys of participants.

Iterate:

If some $M \in \mathcal{M}$ looks like a pair $\langle M_1, M_2 \rangle$ then

- add M_1, M_2 to $\mathcal{M}$;
- for M, record that it is a pair $\langle M_1, M_2 \rangle$.

Translating Computational $\to$ Formal

Consider

- a computational trace,
  - Actually, the set $\mathcal{M}$ of messages appearing in it.
- the set $\mathcal{K}$ of secret decryption keys of participants.

Iterate:
If some $M \in \mathcal{M}$ looks like a symmetric key then

- add M to $\mathcal{K}$;
- pick a new formal symmetric key K and associate it with M.

Concerning symmetric encryption, attention has to be paid to encryption cycles.

Translating Computational $\rightarrow$ Formal

Consider

- a computational trace,
  - Actually, the set $\mathcal{M}$ of messages appearing in it.
- the set $\mathcal{K}$ of secret decryption keys of participants.

Iterate:
If some $M \in \mathcal{M}$ looks like an encryption then try to decrypt it with all keys in $\mathcal{K}$. If $M_0 = \mathcal{D}(M_k, M)$ for some $M_k \in \mathcal{K}$, then

- add M_0 to $\mathcal{M}$;
- for M, record that it is an encryption of M_0 with the formal key corresponding to the encryption key of M_k.

Translating Computational $\rightarrow$ Formal

Consider

- a computational trace,
  - Actually, the set $\mathcal{M}$ of messages appearing in it.
- the set $\mathcal{K}$ of secret decryption keys of participants.

Iterate:
If some $M \in \mathcal{M}$ looks like a signature then try to verify it with all verification keys in $\mathcal{M}$. If $\mathcal{V}(M_k, M)$ is successful, then

- add $M_0 = get\_message(M)$ to $\mathcal{M}$;
- for M, record that it is the signature of M_0 verifiable with the formal key corresponding to M_k.

Translating Computational $\rightarrow$ Formal

Consider

- a computational trace,
  - Actually, the set $\mathcal{M}$ of messages appearing in it.
- the set $\mathcal{K}$ of secret decryption keys of participants.

Iterate:

etc. Try to decompose the messages in $\mathcal{M}$ as much as possible.

Translating Computational $\rightarrow$ Formal

Consider

- a computational trace,
  - Actually, the set $\mathcal{M}$ of messages appearing in it.
- the set $\mathcal{K}$ of secret decryption keys of participants.

In the end:

- for each uninterpreted message in $\mathcal{M}$: associate it with a new formal nonce.
- Construct the formal trace using the structure of messages that we recorded.

Invalid formal trace $\Rightarrow$ broken primitive

If the trace is invalid, then the adversary did one of the following:

- forged a signature;
- guessed a nonce, symmetric key, or signature that it had only seen encrypted.

We run the protocol while using the encryption / signing oracles to encrypt / sign. We guess at which point the break happens.

- We use the oracles for this particular key.
- A forged signature promptly gives us a break of UF-CMA.
- For guessed nonce, key or signature we generate two copies of it and use the messages derived from these two copies as the inputs to the oracle $\mathcal{E}(k, \pi_b(\cdot, \cdot))$.
  - After learning the nonce / key / signature, we learn b.

Trace properties

- A trace property of P is a subset of the set of all formal traces.
- A protocol formally satisfies a trace property P if all its formal traces belong to P.
- A protocol computationally satisfies a trace property P if for almost all computational traces t^c of the protocol there exists a trace $t^f \in P$, such that $t^f \leq t^c$.

Theorem. If a protocol formally satisfies some trace property P, then it also computationally satisfies P.

Confidentiality of nonces

- In the formal setting, the confidentiality of a certain nonce N means that N will not be included in the knowledge set of the adversary.
- In the computational setting, the confidentiality of a certain nonce N means that no PPT adversary $\mathcal{A}$ can guess b from the following:
  - Run the protocol normally, with $\mathcal{A}$ as the adversary, until...
  - $\mathcal{A}$ denotes one of the just started protocol sessions as "under attack".
  - Generate a random bit b and two nonces N_0 and N_1.
  - Use N_b in the attacked session in the place of N.
  - Continue executing the protocol until $\mathcal{A}$ stops it.
  - Give N_0 and N_1 to $\mathcal{A}$.

Theorem. Formal confidentiality of a nonce implies its computational confidentiality.