Continuous After-the-fact Leakage-Resilient eCK-Secure Key Exchange

Janaka Alawatugoda[1], Colin Boyd[2], Douglas Stebila[1]

[1]Queensland University of Technology, Brisbane, Australia
[2]Norwegian University of Science and Technology, Trondheim, Norway

Presentation Outline

Diffie–Hellman-based one-round protocol

long-term key: x_A		long-term key: x_B
A		**B**
$a \leftarrow R$		
$A = g^a$		
	$\xrightarrow{A}$	
		$b \leftarrow R$
		$B = g^{r_B}$
	$\xleftarrow{B}$	
$K = F(x_A, r_A, B, \text{public info})$		$K = \hat{F}(x_B, r_B, A, \text{public info})$

- a and b are the *ephemeral keys*
- K is the *session* key
- Prominent concrete protocols include MQV, HMQV and UM

Key Exchange Security Models

- Designed to capture informal security goals.
- Typical elements in a security model:
  - **Adversary Capabilities** - set of adversary operations. Always allows adversary to view protocol runs and alter/inject messages.
  - **Security Game** - the order in which the adversary operations are performed.
  - **Security Definition** - the requirement to win the security game. Usually require the adversary to reliably distinguish session key from a random string.

Adversarial Capabilities (eCK model)

- Adversary runs the protocol:
  - `Send`: Adversary can send a message to a protocol session which answers according to the specification.
- Adversary compromise certain secret keys:
  1. `SessionKeyReveal`: Adversary is given the session key of a session.
  2. `EphemeralKeyReveal`: Adversary is given the ephemeral key of a session.
  3. `Corrupt`: Adversary is given the long-term secrets of a principal.
- Adversary asks for the challenge:
  - `Test`: Adversary is given either the session key or a random string.

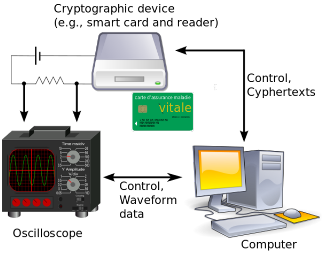
Security Game

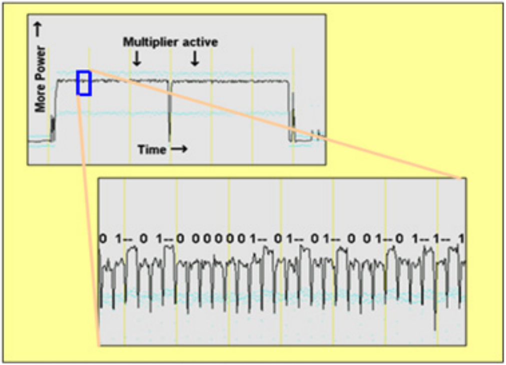
- **Stage 1:** The adversary performs `Send`, `SessionKeyReveal`, `EphemeralKeyReveal` and `Corrupt` operations.
- **Stage 2:** `Test` operation to any uncompromised test-session.
- **Stage 3:** `Send`, `SessionKeyReveal`, `EphemeralKeyReveal` and `Corrupt` keeping the test-session uncompromised.
- **Stage 4:** Adversary outputs a bit as its guess whether the `Test` operation output was random (0) or the real session key (1).

The adversary wins the game if it guesses correctly.

Side-Channel Attacks

- Leaking information from cryptographic implementations can be used as side-channels to reveal secrets.
- Side-channels: Timing information, Power consumption information, Cache-access pattern, EM-radiation etc.

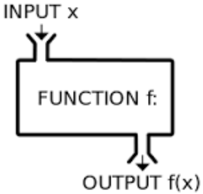


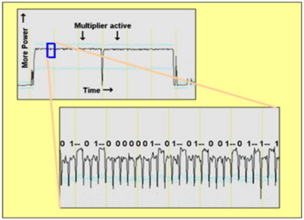


Plots courtesy of Cryptography Research, Inc.

Leakage-Resilient Cryptography

- Provable security against side-channel attacks.
- Constructing cryptographic schemes in leakage-resilient manner.
  - Model leakage.
  - Prove that even in the presence of certain amount of leakage to an attacker, a cryptographic scheme is secure.
- Adversary gets the leakage of the secret x using adversary-chosen, adaptive, efficiently computable leakage functions f.





Plots courtesy of Cryptography Research, Inc.

Modelling Leakage

Different options have been used.

Leakage function f	restricted class of functions (eg hard to invert functions) *or* arbitrary polynomial time functions
Output amount of the f	bounded *or* continuous leakage
When to apply f?	only before the security challenge *or* before + after the security challenge (after-the-fact)

Ideally want the leakage function be arbitrary and allow continuous, after the fact leakage.

Modelling Leakage in the CAFL-eCK Model

- Follow model of Dziembowski and Faust (2011)
- An arbitrary polynomial time leakage function $\mathbf{f}$ is used to model the leakage s.t. $\mathbf{f}(sk) = leakage$.
- Leakage is modelled in a place where computation takes place using long-term secret keys.
- Total leakage amount is unbounded (continuous leakage).
- Allows after-the-fact leakage.

Adversarial Capabilities of the CAFL-eCK Model

- Adversary run the protocol:
  - `Send`$(m, \mathbf{f})$: Models the capabilities of the adversary who can initiate, delay, modify or insert protocol messages m. The adversary observes the leakage of the secret key $\mathbf{f}(sk)$, whenever a computation takes place in a party.
- Adversary compromise certain secret keys:
  1. `SessionKeyReveal`: Adversary is given the session key of a session.
  2. `EphemeralKeyReveal`: Adversary is given the ephemeral keys (per-session randomness) of a session.
  3. `Corrupt`: Adversary is given the long-term secrets of a principal.
- Adversary asks for the challenge:
  - `Test`: Adversary is given either the real or a random session key.

Comparison with Earlier Models

Security model	`Ephemeral Key`	`Long-term Key`	**Combinations**	**Leakage resilience**
eCK (2007)	Yes	Yes	4/4	None
MO (2011)	Yes	Yes	4/4	Bounded, before-the-fact
BAFL-eCK (2014)	Yes	Yes	4/4	Bounded, after-the-fact
CAFL (2014)	Yes	Yes	2/4	Continuous, after-the-fact
CAFL-eCK (now)	Yes	Yes	4/4	Continuous, after-the-fact

1. $\texttt{Corrupt}(U)$ and $\texttt{Corrupt}(V)$.
2. $\texttt{Corrupt}(U)$ and $\texttt{EphemeralKeyReveal}(V, U, s)$.
3. $\texttt{Corrupt}(V)$ and $\texttt{EphemeralKeyReveal}(U, V, s')$.
4. $\texttt{EphemeralKeyReveal}(V, U, s)$ and $\texttt{EphemeralKeyReveal}(U, V, s')$.

Dziembowski–Faust leakage-resilient storage scheme

For any $n \in \mathbb{N}$, the storage scheme $(\mathrm{Encode}, \mathrm{Decode})$ efficiently stores an element $s \in \mathbb{Z}_q^*$ where:

- $\mathrm{Encode}(s)$: $s_L \xleftarrow{\$} (\mathbb{Z}_q^*)^n \backslash \{(0^n)\}$, then $s_R \leftarrow (\mathbb{Z}_q^*)^n$ such that $s_L \cdot s_R = s$ and outputs (s_L, s_R).
- $\mathrm{Decode}(s_L, s_R)$: outputs $s_L \cdot s_R$.

Dziembowski–Faust leakage-resilient storage scheme

For any $n \in \mathbb{N}$, the storage scheme $(\mathrm{Encode}, \mathrm{Decode})$ efficiently stores an element $s \in \mathbb{Z}_q^*$ where:

- $\mathrm{Encode}(s)$: $s_L \xleftarrow{\$} (\mathbb{Z}_q^*)^n \backslash \{(0^n)\}$, then $s_R \leftarrow (\mathbb{Z}_q^*)^n$ such that $s_L \cdot s_R = s$ and outputs (s_L, s_R).
- $\mathrm{Decode}(s_L, s_R)$: outputs $s_L \cdot s_R$.

The values (s_L, s_R) can then be *refreshed* using the following algorithm

- Refreshing s_R:
  1. Choose $A, B \in (\mathbb{Z}_q^*)^n$ such that $A \cdot B = 0^m$.
  2. Choose $M \in (\mathbb{Z}_q^*)^{n \times n}$ such that $s_L \cdot M = A$.
  3. $s_{R'} = R + M \cdot B$.
- Refreshing s_L:
  1. Choose $\tilde{A}, \tilde{B} \in (\mathbb{Z}_q^*)^n$ such that $\tilde{A} \cdot \tilde{B} = 0^m$.
  2. Choose $\tilde{M} \in (\mathbb{Z}_q^*)^{n \times n}$ such that $\tilde{M} \cdot s_{R'} = \tilde{B}$.
  3. $s_{L'} = L + \tilde{A} \cdot \tilde{M}$.

Constructing an CAFL-eCK-secure Protocol

- Construct a simple eCK-secure protocol
- Use leakage-resilient storage scheme and its refreshing protocol to convert to a CAFL-eCK-secure protocol.
- If the storage scheme and its refreshing protocol are leakage-resilient, the protocol is CAFL-eCK-secure. Allows 15% continuous leakage of the secret key with $n = 21$.

An eCK-secure Protocol

A		**B**
	Initial Setup	
$a \xleftarrow{\$} \mathbb{Z}_q^*, A \leftarrow g^a$		$b \xleftarrow{\$} \mathbb{Z}_q^*, B \leftarrow g^b$
	Message Exchange	
$x \leftarrow \mathbb{Z}_q^*, X \leftarrow g^x$	$\xrightarrow{ID_A, X}$	$y \xleftarrow{\$} \mathbb{Z}_q^*, Y \leftarrow g^y$
	$\xleftarrow{ID_B, Y}$	
	Session Key Computation	
$Z_1 \leftarrow B^a, Z_2 \leftarrow B^x$		$Z_1' \leftarrow A^b, Z_2' \leftarrow X^b$
$Z_3 \leftarrow Y^a, Z_4 \leftarrow Y^x$		$Z_3' \leftarrow A^y, Z_4' \leftarrow X^y$
$sid_A = ID_A, X, ID_B, Y$		$sid_B = ID_A, X, ID_B, Y$
$K \leftarrow \mathrm{H}(Z_1, Z_2, Z_3, Z_4, sid_A)$		$K \leftarrow \mathrm{H}(Z_1', Z_2', Z_3', Z_4', sid_B)$
	K is session key	

An eCK-secure Protocol

A		**B**
	Initial Setup	
$a \xleftarrow{\$} \mathbb{Z}_q^*, A \leftarrow g^a$		$b \xleftarrow{\$} \mathbb{Z}_q^*, B \leftarrow g^b$
	Message Exchange	
$x \leftarrow \mathbb{Z}_q^*, X \leftarrow g^x$	$\xrightarrow{ID_A, X}$	$y \xleftarrow{\$} \mathbb{Z}_q^*, Y \leftarrow g^y$
	$\xleftarrow{ID_B, Y}$	
	Session Key Computation	
$\boxed{Z_1 \leftarrow B^a}, Z_2 \leftarrow B^x$		$\boxed{Z_1' \leftarrow A^b, Z_2' \leftarrow X^b}$
$\boxed{Z_3 \leftarrow Y^a}, Z_4 \leftarrow Y^x$		$Z_3' \leftarrow A^y, Z_4' \leftarrow X^y$
$sid_A = ID_A, X, ID_B, Y$		$sid_B = ID_A, X, ID_B, Y$
$K \leftarrow \mathrm{H}(Z_1, Z_2, Z_3, Z_4, sid_A)$		$K \leftarrow \mathrm{H}(Z_1', Z_2', Z_3', Z_4', sid_B)$
	K is session key	

Using LR-stored secrets for exponentiation

- Let $s \in \mathbb{Z}_q^*$ be a long-term secret key and $E = g^e$ be a received ephemeral value. Then, the value $Z = E^s$ needs to be computed.
- The secret key is encoded as s_L, s_R. So the vectors $s_L = (s_{L1}, \cdots, s_{Ln})$ and $s_R = (s_{R1}, \cdots, s_{Rn})$ are such that $s = s_{L1}s_{R1} + \cdots + s_{Ln}s_{Rn}$.
- The computation of E^s can be performed as two component-wise computations:
  - compute the intermediate vector $T = (E^{s_{L1}}, \cdots, E^{s_{Ln}})$
  - compute the element $Z = E^{s_{L1}s_{R1}} E^{s_{L2}s_{R2}} \cdots E^{s_{L1}s_{R1}} = E^{s_{L1}s_{R1} + \cdots + s_{Ln}s_{Rn}} = E^s$.

An CAFL-eCK-secure Protocol

A		B
Initial Setup		
$a \xleftarrow{\$} \mathbb{Z}_q^*, A \leftarrow g^a$		$b \xleftarrow{\$} \mathbb{Z}_q^*, B \leftarrow g^b$
$(a_L^0, a_R^0) \leftarrow \mathrm{Encode}(a)$		$(b_L^0, b_R^0) \leftarrow \mathrm{Encode}(b)$
Erase a		Erase b
Message Exchange		
$x \leftarrow \mathbb{Z}_q^*, X \leftarrow g^x$	$\xrightarrow{ID_A, X}$	$y \xleftarrow{\$} \mathbb{Z}_q^*, Y \leftarrow g^y$
	$\xleftarrow{ID_B, Y}$	
Session Key Computation		
$T_1 \leftarrow B^{a_L^j}, Z_1 \leftarrow T_1^{a_R^j}$		$T_1' \leftarrow A^{b_L^j}, Z_1' \leftarrow (T_1')^{b_R^j}$
$T_3 \leftarrow Y^{a_L^j}, Z_3 \leftarrow T_2^{a_R^j}$		$T_2' \leftarrow X^{b_L^j}, Z_2' \leftarrow (T_2')^{b_R^j}$
$Z_2 \leftarrow B^x, Z_4 \leftarrow Y^x$		$Z_3' \leftarrow A^y, Z_4' \leftarrow X^y$
$sid_A = ID_A, X, ID_B, Y$		$sid_B = ID_A, X, ID_B, Y$
$K \leftarrow \mathrm{H}(Z_1, Z_2, Z_3, Z_4, sid_A)$		$K \leftarrow \mathrm{H}(Z_1', Z_2', Z_3', Z_4', sid_B)$

Summary

- First concrete construction of strongly secure key exchange with continuous after-the-fact leakage resilience.
- Possible improvements:
    - increase efficiency with regard to randomness and key computation;
    - different leakage resilience models;
    - standard model.