Leakage-Resilient Cryptography
-- for symmetric primitives
How to construct cryptodevices?

**very secure**
- well-defined mathematical object
- often proof-driven security analysis

Leakage Resilient Crypto
Extend concept of proof-driven security analysis to implementation-level

**much less secure!**
- many ways of implementing: details matter!
- security analysis by experiments, rarely proofs
The approach of provable security

1. Define model & security notion

Example: Digital signatures

message → key K → signature
1. Define model & security notion

Example: Digital signatures

Scheme is secure: no adversary can output a valid forgery!
The approach of provable security

1. Define model & security notion

2. Design cryptoscheme
   Usually described in mathematical language
The approach of provable security

1. Define model & security notion

2. Design cryptoscheme
   Usually described in mathematical language

3. Prove security
   Reduce security of complex scheme to **simple** assumption, e.g.,
   - Number theory: studied intensively in math
   - One-wayness of function: major breakthrough in complexity

shows security not only against **one specific attack**, but any efficient (PPT) attack within the model (if assumption holds)
Time to relax?

Security proof implies…
- secure against all known attacks
- secure against all attacks that may be discovered in future

Provably secure systems get broken in practice!
Underlying assumptions are false? Not for standard assumptions
Bugs in proofs? Only rarely!

So what’s wrong?
Models make idealized assumptions

- Hash functions behave as random oracles
- Black-box computation
Black-box model vs. Reality

Security model: Black box
Attacking mathematical algorithm

input $X$ → key $K$ → output $Y$
Controls inputs/outputs
But: Internal computation and key completely hidden

Reality:
Attacking the implementation
Implementations leak partial information about internals
Leakage: e.g., power consumption, running time, electromagnetic radiation…
Physical devices are not black boxes

1. Proofs in black-box model less meaningful
2. Even worse: Side-channel attacks exploit leakage and break assumptions

Goal of leakage-resilient crypto

Weaken black-box assumption and incorporate broad classes of leakage into model

Develop new cryptoalgorithms with built-in resistance against leakage and prove security

Important question: what are these classes?
Leakage Resilient Cryptography

Hot topic…

Digital signatures: [AWD09, KV09, FKPR10, DHLW10, BKKV10, BSW11,…]

Public key encryption: [AGV09, NS09, DHLW10, BKKV10, BSW11,…]

Identity based encryption: [DHLW10, CDRW10, LRW11,…]

Multiparty Computation: [ISW03, FRRTV10, GR10, JV10,…]

Zero Knowledge: [GJS11,…]

But surprisingly little is known about symmetric primitives…

Pseudorandom Generators: [DP08, Pie09, YSPY10]

Pseudorandom Functions & Permutations: [DP10, FPS11]

Most of this talk
Defining leakage

Modeled by a leakage function $f$
Adversary obtains leakage $f(K)$

Arbitrary leakage function? No…
$\Rightarrow$ e.g.: $f(K) = K$ means no security
Some restrictions are necessary
Does this make sense in practice?

Arbitrary efficient adversary
Defining leakage

Modeled by a leakage function $f$
Adversary obtains leakage $f(K)$

Arbitrary leakage function? No…
- e.g.: $f(K) = K$ means no security

Some restrictions are necessary

Does this make sense in practice?
In many cases yes…

Power consumption modeled by $f(K) =$ Hamming weight of wires in circuit
Running time of device
What are possible restrictions?

One attempt: consider specific leakage function

But we do not want to protect only against specific attacks

(such as: Hamming weight, timing)

Leakage Resilient Crypto: consider broad classes of leakage functions!
A broad class of leakage functions

L is class of poly-time computable input shrinking functions
L = \{ f : \{0,1\}^m \rightarrow \{0,1\}^n \}, with n < m

**Observation:** f is poly-time \(\Rightarrow\) can simulate all intermediate values & leak about them

Many realistic leakages: HW, running time exploit only poly-log amount of information

**Problem:** total leakage \(<<\) length of the key

**Reality:** Many observations are possible (many attacks exploit a large number of observations)
Continuous Leakage Model

Many adaptive observations:

\[ X_1 f_1(\mathbf{K}) Y_1 \]

\[ \cdots \]

\[ X_q f_q(\mathbf{K}) Y_q \]
Continuous Leakage Model

Many adaptive observations:

\[ f_1(K), f_2(K), \ldots, f_q(K) \]

Bounded per observation to \( n \) bits
But: total leakage \( \gg |K| \)

Models, e.g., DPA where we need many power samples to recover the key
1. Leakage Resilient Stream Cipher
2. Leakage Resilient PRFs
3. Leakage Resilient Circuits
Leakage Resilient Stream Cipher

First construction: Dziembowski-Pietrzak-08
Simpler construction: Pietrzak-09

stream ciphers ≈ pseudorandom generators

short key

long pseudo random stream $X$

Pseudorandomness: no efficient (PPT) adversary can distinguish $X$ from random
Stream ciphers in practice

Stream $X$ is generated in rounds from $K$ (one block per round)
Standard Security Notion

Given previous blocks, next block should look random

How to extend to leakage setting?
Given previous blocks, next block should look random

Adversary knows also leakage

Should look random

Poly-time computable bounded-output leakage function
Standard Security Notion

Given previous blocks, next block should look random

SC

Given previous blocks, next block should look random

\[ K \]

Adversary knows also leakage

Some problems?

1. Adversary can learn entire key \( K \) bit-by-bit

2. Given leakage \( f_{i-1}(K) \), the block \( X_i \) is not pseudorandom anymore

\[ \Rightarrow f_{i-1}(K) \text{ can leak some bits about } X_i \]
Key evolution

In each round key $K_i$ is used to compute new state $K_{i+1}$

- Requirement: Key evolution must be deterministic!
Otherwise it cannot be used for encryption!
Key evolution

In each round key $K_i$ is used to compute new state $K_{i+1}$

$K_1 \rightarrow K_2 \rightarrow K_3 \ldots$

- Requirement: Key evolution must be deterministic!
  Otherwise it cannot be used for encryption!
- Also key update leaks!

Is key evolution sufficient?
Is key evolution sufficient?

Learning key bit-by-bit does not work anymore 😊

\[ K_1 \rightarrow K_2 \rightarrow K_3 \ldots \]

Can \( X_2 \) be pseudorandom given leakage \( f_1(K_1) \)? No! 😞

Key evolution deterministic: \( f_1 \) computes \( K_2 \) and leaks bits of \( X_2 \)

Even worse: pre-computation attack

Leakage function \( f_1 \ldots f_{i-1} \) leak from future state \( K_i \)

⇒ may reveal entire \( K_i \) even with one bit of leakage
How to avoid this attack?

Pre-computation attack relevant in practice? No!

It’s a problem of the model…

Use restriction introduced by Micali-Reyzin-04:

“only computation leaks information”

or in other words:

“untouched memory cells do not leak information”
Only computation leaks

state
Only computation leaks

state: divided into parts

L

R
Only computation leaks

state: divided into parts

if used in current computation
⇒ $f(L)$ leaks to adversary

if not accessed:
⇒ does not leak

Restriction can be relaxed in many cases…
Independent leakages

State: divided into parts

\[ \text{if used in current computation} \implies f(L) \text{ leaks to adversary} \]
\[ \text{if not accessed:} \implies f(R) \text{ leaks (independently of L)} \]

How can we use this to avoid pre-computation?
Divide memory into three parts: L,X,R

- **L** holds pseudorandom output of the cipher
The stream cipher – high-level view

Divide memory into three parts: L,X,R

- L: Holds secret state
- X: Intermediate state
- R: Output state
The stream cipher – high-level view

Divide memory into three parts: L,X,R

\[
\begin{align*}
L_1 & \quad L_2 := L_1 \\
X_1 & \quad X_2 \\
R_1 & \quad R_2
\end{align*}
\]

unmodified

SC
Divide memory into three parts: L, X, R

\[ L_2 := L_1 \]

\[ f(X_1, R_1) \]

unmodified
The stream cipher – high-level view

Divide memory into three parts: L, X, R

- \( L_1 \)
- \( L_2 := L_1 \)
- \( X_1 \)
- \( X_2 \)
- \( R_1 \)
- \( R_2 \)

Recall: leakage is polynomial-time computable function, i.e., we can also leak from \((X_2, R_2)\)
Divide memory into three parts: L,X,R

Alternation prevents pre-computation attack
E.g.: $f_1$ cannot leak about state $(L_3,X_3,R_3)$
The stream cipher – high-level view

Divide memory into three parts: L,X,R

What can we prove?

X_i is pseudorandom given X_1,...,X_{i-1} and leakages f_1(X_1,R_1)...f_{i-2}(X_{i-2},L_{i-2})
The stream cipher – high-level view

Divide memory into three parts: L,X,R

L₁ | X₁ | R₁
unmodified

L₂ | X₂ | R₂
SC
unmodified

L₃ | X₃ | R₃
SC
unmodified

L₄ | X₄ | R₄

How can we initialize SC?
Use **randomness extractor**: generates from short random seed $X_{i-1}$ and high min-entropy source $R_{i-1}$ an almost uniform string $Y_i$

**But**: $Y_i$ is much shorter than evolved state $R_i$ and output $X_i$

Use **pseudorandom generator**: generates from short random seed long pseudorandom string $(X_i \, R_i)$ as good as uniform

**Security proof**: see the paper!
Alternative Instantiations

Pietrzak-2009: use a weak PRF $F$ (for fixed key and random inputs, the output is pseudorandom)

$$L_{i-1} \quad X_{i-1} \quad R_{i-1}$$

$$(X_i, Y_i) = F(R_{i-1}, X_{i-1})$$

Yu-Standaert-Pereira-Yung-2010:

• even simpler construction & tight security reduction

• But in the Random Oracle model $\Rightarrow$ leakage function cannot query the RO
1. Leakage Resilient Stream Cipher
2. Leakage Resilient PRFs
3. Leakage Resilient Circuits
Pseudorandom Functions

Pseudorandom Generator $G(K)$: for short key $K$ outputs long pseudorandom string $X$

\[ K \xrightarrow{G} X \]

Pseudorandom Function $F(K,.)$: for short key $K$ can be queried on input $X$ and outputs pseudorandom string $Y$

\[ K \xrightarrow{F} (X) \xrightarrow{Y} \]
Pseudorandom Functions

Pseudorandom Generator $G(K)$: for short key $K$ outputs long pseudorandom string $X$

Pseudorandom Function $F(K,\cdot)$: for short key $K$ can be queried on input $X$ and outputs pseudorandom string $Y$

Behaves as function: for same input, it returns the same output

Standard security notion: $Y_{i+1}$ is pseudorandom given $Y_1, \ldots, Y_i$, if $X_{i+1}$ has not been queried

How can we extend this to leaky setting?
How to extend to leaky setting?

\[ Y_{q+1} \text{ is pseudorandom if } X_{q+1} \text{ has not been queried yet} \]

**Problem:** Leakage allows to recover \( K \) bit-by-bit.

**Can we use again key evolution?** No: For two identical queries PRF has to return same values!
We use the following restrictions:

1. Leakage is bounded per observation
2. Only computation leaks information
   
   But: at lower architectural level: computation of PRF is structured into t time steps which leak independently
3. Leakage functions are fixed a-priori by the device
   Reasonable in reality: adversary has no full adaptive control over functions
Leakage Resilient PRF

Standard way to build PRF is via GGM-tree construction

\[ G \]

\[ \vdots \]

\[ G \]

\[ \vdots \]

\[ G \]

\[ \vdots \]

\[ \vdots \]
Is GGM leakage resilient?

Each node leaks independently & leakage functions are fixed a-priori

Does this suffice? No: pre-computation attack still possible

Dodis-Pietrzak-10: hybrid of a leakage resilient stream cipher and the GGM tree is a leakage resilient PRF

F-Pietrzak-Schipper-11: simpler & more natural construction (only secure for non-adaptive input queries)
1. Leakage Resilient Stream Cipher
2. Leakage Resilient PRFs
3. Leakage Resilient Circuits
Proof of leakage resilient AES?

**Unlikely:** we cannot prove that AES is black-box secure

**Idea:** show that implementation is as secure as in bb-world

**Leakage Resilient Circuit Compilers**

Arbitrary Boolean circuit, e.g., AES
Unlikely: we cannot prove that AES is black-box secure.

Idea: show that implementation is as secure as in bb-world.

**Leakage Resilient Circuit Compilers**

**Circuit compiler:**
**Input:** description of circuit $C$ and key $K$
**Output:** description of transformed circuit $C'$ and key $K'$
Leakage Resilient Circuits

Unlikely: we cannot prove that AES is black-box secure

Idea: show that implementation is as secure as in bb-world

Leakage Resilient Circuit Compilers

Circuit compiler:
Input: description of circuit $C$ and key $K$
Output: description of transformed circuit $C'$ and key $K'$

Transformed circuit $C'$:
resistant to continuous leakages from some function class $L$

$\Rightarrow$ Even given leakage $C'$ is as secure as in bb-world
What is the class of functions $L$?

**Theorem 1:** A compiler that makes any circuit resilient to probing up to $t$ wires [Ishai, Sahai, Wagner 03].

$L$ is specific leakage function that allows the adversary to learn the value of up to $t$ wires.
What is the class of functions $L$?

**Theorem 1:** A compiler that makes any circuit resilient to probing up to $t$ wires

**Theorem 2:** A compiler that makes any circuit resilient to global computationally weak leakages

Leakage functions not PPT, but from weak complexity class: cannot compute certain linear functions, e.g., parity

$\Rightarrow$ class of leakage functions $L = AC^0$
What is the class of functions L?

**Theorem 1:** A compiler that makes any circuit resilient to probing up to $t$ wires

Leakage is \{wire$_i$ + noise $\eta_i$\}

**Theorem 2:** A compiler that makes any circuit resilient to global computationally weak leakages

[F-Rabin-Reyzin-Tromer-Vaikuntanathan-10]

**Theorem 3:** A compiler that makes any circuit resilient to global noisy leakages

[F-Rabin-Reyzin-Tromer-Vaikuntanathan-10]

Proof-driven analysis of masking-based countermeasure

Can we get circuit compilers for broader classes?
Circuit compilers for PPT leakage?

Juma-Vahlis-2010: uses fully homomorphic encryption
Goldreich-Rothblum-2010: encrypts every wire of original circuit with a fresh pk/sk

⇒ Both are impractical!

Can we do better?

Dziembowsk-F-11: using two source extractors
It’s provable secure, but does this offer better real world security than standard masking?
We are currently exploring this with practitioners!
Yes, extending the black box model is possible

More interaction between theoreticians and practitioners is needed to find valid restrictions and efficient schemes

Many open problems, e.g.,

• Leakage resilient block-ciphers
• Security against continuous hard-to-invert leakage
• More results for computationally bounded leakage
Thank you!