Leakage Resilient Computation

Yevgeniy Vahlis
joint work with Ali Juma, Charles Rackoff
Crypto

Encryption Box
Crypto

Encryption Box

Message $M_0$
Message $M_1$
Crypto

\[ b \in_r \{0, 1\} \]

Encryption Box

Message \( M_0 \)

Message \( M_1 \)
Crypto

\[ b \in_r \{0, 1\} \rightarrow \text{Encryption Box} \]

Encryption of \( M_b \)
Crypto

\[ b \in_r \{0, 1\} \]

Encryption Box

Encryption of \( M_b \)

\[ b=? \]
Crypto in Real Life

RSA

\( N = pq \)
Crypto in Real Life

RSA
N = pq

Message \( M_0 \)
Message \( M_1 \)
Crypto in Real Life

\[ b \in_r \{0, 1\} \rightarrow \text{RSA} \]

\[ N = pq \]

Message \( M_0 \)
Message \( M_1 \)
Crypto in Real Life

\[ b \in_r \{0, 1\} \]

RSA
\[ N=pq \]

Encryption of \( M_b \)
Crypto in Real Life

\[ b \in_r \{0, 1\} \rightarrow \text{RSA} \]
\[ N = pq \]

Encryption of \( M_b \)
Crypto in Real Life

$b \in_r \{0,1\} \rightarrow \text{RSA} \quad N=pq$

Encryption of $M_b$
Crypto in Real Life

$b \in_r \{0, 1\}$

$\text{RSA}$

$N = pq$

Encryption of $M_b$
Crypto in Real Life

\[ b \in_r \{0, 1\} \]

RSA
\[ N = pq \]

Encryption of \( M_b \)
Crypto in Real Life

\[ b \in_r \{0, 1\} \]

Encryption of \( M_b \)
Crypto in Real Life

\[ b \in_r \{0, 1\} \]

Encryption of \( M_b \)

\[ b! \]
Crypto in Real Life

N = pq
Crypto in Real Life

Solutions:
Crypto in Real Life

Solutions:
• Heat isolation
Crypto in Real Life

Solutions:
• Heat isolation
• Uniform cooling and quite fans
Crypto in Real Life

Solutions:
- Heat isolation
- Uniform cooling and quite fans
- Electromagnetic isolation
Crypto in Real Life

Solutions:
- Heat isolation
- Uniform cooling and quite fans
- Electromagnetic isolation
- ???
Algorithmic Protection

N = pq

RSA
Algorithmic Protection

In recent years:
In recent years:
• Reduce assumptions about hardware
In recent years:
- Reduce assumptions about hardware
- Design algorithms that are secure under leakage
Previous Work

• [Goldreich & Ostrovsky 96] protect against complete leakage of memory when CPU is secure.
• [Ishai & Sahai & Wagner 2003] assume adversary leaks value of a fixed number of wires
• [Micali & Reyzin 04] introduce axioms and framework
• [Goldwasser & Kalai & Rothblum 2008] one-time programs
• [Dziembowski & Pietrzak 2008] Leakage resilient stream cipher in the split state model
• [Faust & Reyzin & Tromer 2009] protect against AC0 leakage functions (needs secure component)
Our Results
Our Results

- A compiler that transforms any keyed primitive $H_K$ into a stateful algorithm $G_{\text{state}}$
- $G_{\text{state}}$ is resilient against length bounded leakage in each invocation
- Need a fixed size, memory-less secure component
Our Results

- A compiler that transforms any keyed primitive $H_K$ into a stateful algorithm $G_{state}$
- $G_{state}$ is resilient against length bounded leakage in each invocation
- Need a fixed size, memory-less secure component

Achieve leak-resilience for arbitrary complexity from leak-resilience for fixed complexity
Model

Want to protect $H_K(x)$

Two computers that communicate over a public channel. Initialization is secure.
Model

Want to protect $H_K(x)$

Two computers that communicate over a public channel. Initialization is secure.
Model

Want to protect $H_K(x)$

Two computers that communicate over a public channel. Initialization is secure.
Evaluating $H_K(x)$
Model

Evaluating $H_K(x)$

- CPU A
  - Mem$A_i$

- CPU B
  - Mem$B_i$
Model

Evaluating $H_K(x)$

CPU A

MemA$_i$

CPU B

MemB$_i$

$x$
Model

Evaluating $H_K(x)$

CPU A

MemA$_i$

MemA$_{i+1}$

CPU B

MemB$_i$

MemB$_{i+1}$

$H_K(x)$

x
Leakage

$R_A$ \downarrow 

\begin{array}{|c|}
\hline 
CPU A \\
\hline 
MemA_1 \\
\hline 
\end{array} 

$R_B$ \downarrow 

\begin{array}{|c|}
\hline 
CPU B \\
\hline 
MemB_1 \\
\hline 
\end{array}
Leakage

CPU A

MemA₁

CPU B

MemB₁

Rₐ

Rₜ

f₁

x₁
Leakage

CPU A

MemA₁
Computing...

CPU B

MemB₁

$R_A$

$R_B$

$f_1$

$x_1$
Leakage

Computing... $f_1(MemA_1, R_A) = L_1$
Leakage

$R_A$ 
\[ \text{CPU A} \\ \text{Mem}_{A_1} \]

$R_B$ 
\[ \text{CPU B} \\ \text{Mem}_{B_1} \]

$x, L_1$
Leakage

CPU A
MemA₁

CPU B
MemB₁

RA

RB

x, L₁
Leakage

CPU A
MemA₁

CPU B
MemB₁

Rₐ

x, L₁

Rₜ

f₂
Leakage

CPU A

MemA₁

CPU B

MemB₁

Computing...

\( f_2 \)

\( x, L_1 \)

\( R_A \)

\( R_B \)
Leakage

CPU A

MemA₁

CPU B

MemB₁

Computing...

Rₐ

Rₜ

x, L₁

f₂

z
Leakage

CPU A

MemA₁

CPU B

MemB₁

Computing...

\[ L₂ = f₂(\text{MemB}_₁, R_B, z) \]

\[ f₂ \]

\[ x, L₁ \]

\[ z \]
Leakage

CPU A

MemA₁

CPU B

MemB₁

RA

RB

x, L₁
Leakage

CPU A

MemA₁

CPU B

MemB₁

help
Leakage

CPU A

MemA₁

CPU B

MemB₁

x, L₁, L₂, L₃...
Leakage

$H_K(x)$

$x, L_1, L_2, L_3, \ldots$
Leakage

Our construction needs 2 flows

$h(x)$

$x, L_1, L_2, L_3, \ldots$
Definition of Security
Definition of Security

Real World
Definition of Security

Real World

\[ x, \text{leakage}(\text{ }) \]
Definition of Security

$x, \text{leakage}(\cdot)$

$H_K(x), \text{leakage}(\text{state})$

Real World
Definition of Security

Real World

Ideal World

\[ x, \text{leakage}(\ ) \]

\[ H_K(x), \text{leakage(state)} \]

Simulator

\[ H_K \]
Definition of Security

Real World

\[ x, \text{leakage}(\cdot) \]
\[ H_K(x), \text{leakage}(\text{state}) \]

Ideal World

\[ H_K \]

Simulator

Definition of Security
Definition of Security

Real World

$x, \text{leakage}(\ )$

$H_K(x), \text{leakage}(\text{state})$

Ideal World

$x$

$H_K(x)$

Fake Leakage

Simulator

$H_K$
Main Tool

Fully Homomorphic Encryption (FHE)

First construction by Gentry at STOC 09
Based on Ideal Lattices

Other restricted constructions are known
[Boneh Goh Nissim 2005]
[Melchor Gaborit Herranz 2008]
Homomorphic Encryption

Regular Public Key Encryption

Generate Keys → pub → Encrypt → C(M) → Decrypt → M

Generate Keys

Encrypt

Decrypt

pub

C(M)

M

pri
Homomorphic Encryption

Regular Public Key Encryption

Generate Keys $\xrightarrow{\text{pub}}$ Encrypt $\xrightarrow{C(M)}$ Decrypt $\xrightarrow{M}$

Homomorphic Encryption
Homomorphic Encryption

Regular Public Key Encryption

Generate Keys → \textit{pub} → Encrypt → \textit{C(M)} → Decrypt → M

Homomorphic Encryption

Evaluate → \textit{C_f(f(M))} → \textit{f( )}
Homomorphic Encryption

Regular Public Key Encryption

Generate Keys → pub → Encrypt → C(M) → Decrypt → M

Homomorphic Encryption

Randomize → C_{f}(f(M)) → Evaluate → C(f(M)) → f( )
Our Construction

Initialization($K$):
Generate keys $pri,pub$
$C(K) = Enc_{pub}(K)$

CPU A

CPU B
Our Construction

Initialization($K$):
- Generate keys $pri, pub$
- $C(K) = Enc_{pub}(K)$

Mem$A_1 = pri$

Mem$B_1 = C(K)$

CPU A

CPU B
First Attempt

Step 1: CPU A

\[ \text{MemA}_i = \text{pri}_i \]

Generate Keys

Encrypt

\[ R_{\text{gen}} \]

\[ R_{\text{enc}} \]
First Attempt

Step 1: CPU A

\[ MemA_i = pri_i \]

Generate Keys

\[ pub_{i+1} \]

Encrypt

\[ R_{enc} \]

\[ R_{gen} \]
First Attempt

Step 1: CPU A

$\text{MemA}_i = \text{pri}_i$

$\text{R}_{\text{enc}}$

$\text{Generate Keys}$

$\text{R}_{\text{gen}}$

$\text{Encrypt}$

$C(\text{pri}_i)$

$\text{pub}_{i+1}$
First Attempt

Step 1: CPU A

MemA_i = pri_i

Generate Keys

Encrypt

C(pri_i)

Send to CPU B
First Attempt

Step 1: CPU A

MemA

= \text{pri}_{i+1}

\text{Generate Keys}

\text{Encrypt}

C(\text{pri}_i)

Send to CPU B
First Attempt

Step 1: CPU A

Mem\textsubscript{A\textsubscript{i}} = \textit{pri}_{i+1}

\begin{align*}
\text{Generate Keys} & \quad \text{Generate Keys} \\
\text{Encrypt} & \quad \text{Encrypt} \\
\text{Send to CPU B} & \quad \text{Send to CPU B}
\end{align*}
First Attempt

Step 2: CPU B

Input: \( x \)

\[ C_{i+1}(pri_i) \]

Evaluate relative to \( pub_{i+1} \)

\[ \text{Decrypt} \]

\[ H \]

\[ \text{MemB}_i = C_i(K) \]
First Attempt

Step 2: CPU B

Input: $x$

$C_{i+1}(pri_i)$

$Mem_{B_i} = C_i(K)$

Evaluate relative to $pub_{i+1}$
First Attempt

Step 2: CPU B

Input: $x$

$C_{i+1}(pri_i)$

$\text{Mem}_{B_i} = C_i(K)$

Evaluate relative to $pub_{i+1}$

Decrypt

$H$
First Attempt

Step 2: CPU B

Input: $x$

$C_{i+1}(pri_i)$

Evaluate relative to $pub_{i+1}$

$Mem_{B_i} = C_i(K)$

Decrypt

$K$

$H$
First Attempt

Step 2: CPU B

Input: $x$

Evaluate relative to $pub_{i+1}$

$C_{i+1}(pri_i)$

$pri_i$ → Decrypt → $K$ → $H$ → $C_{i+1}(H_K(x))$

$MemB_i = C_i(K)$

$C_{i+1}(K)$
First Attempt

Step 2: CPU B

Input: $x$

$C_{i+1}(pri_i)$

Evaluate relative to $pub_{i+1}$

$C_{i+1}(H_{K}(x))$

$Mem_{B_{i+1}} = C_{i+1}(K)$

$C_{i+1}(K)$

$K$

Decrypt

$H$
First Attempt

Step 3: CPU A

$\text{MemA}_i = \text{pri}_{i+1}$

$C_{i+1}(H_K(x))$ → Decrypt
First Attempt

Step 3: CPU A

\[ \text{MemA}_i = \text{pri}_{i+1} \]

\[ C_{i+1}(H_K(x)) \]

\[ H_K(x) \]
First Attempt

Step 3: CPU A

$\text{Mem}A_i = pri_{i+1}$

$C_{i+1}(H_K(x))$ → Decrypt → $H_K(x)$

Looks great! What goes wrong?
First Attempt

Step 3: CPU A

$\text{MemA}_i = \text{pri}_{i+1}$

$C_{i+1}(H_K(x))$ carries history
May contain K

$H_K(x)$

Looks great!
What goes wrong?
First Attempt

Step 3: CPU A

$\text{MemA}_i = \textbf{pri}_{i+1}$

Looks great!
What goes wrong?

$C_{i+1}(H_K(x))$ carries history
May contain $K$

$C_{i+1}(H_K(x))$ flows into Decrypt'

$H_K(x)$ flows into Decrypt'

$K$ flows out of Decrypt'
Second Attempt

Step 2’: CPU B

Input: x

$C_{i+1}(pri_i)$

Evaluate relative to $pub_{i+1}$

Decrypt

$pri_i$

$K$

$H$

$M_{ma}B_{mi}=C_i(KK)$

$C_{i+1}(K)$
Second Attempt

Step 2’: CPU B

Evaluate relative to \( pub_{i+1} \)

Input: \( x \)

\( C_{i+1}(pri_i) \)

\( pri_i \)

Decrypt

\( K \)

\( H \)

Randomize

\( C_{i+1}(H_K(x)) \)

\( MemB_{i+1} = C_{i+1}(K)(K) \)

\( C_{i+1}(K) \)
Second Attempt

Step 2': CPU B

Evaluate relative to $pub_{i+1}$

Input: $x$

$C_{i+1}(pri_i)$

$C_{i+1}(H_K(x))$

$M_i \oplus B_{i+1} = C_i(K)K$

$C_{i+1}(K)$

Randomize

Decrypt

$K$

$H$

$pri_i$
Second Attempt

Step 3: CPU A

\[ \text{MemA}_i = \text{pri}_{i+1} \]

This time \( C_{i+1}(H_K(x)) \) only contains \( H_K(x) \)

\[ C_{i+1}(H_K(x)) \]

\[ H_K(x) \]
Second Attempt

Step 3: CPU A

\[ \text{MemA}_i = \text{pri}_{i+1} \]

This time \( C_{i+1}(H_K(x)) \) only contains \( H_K(x) \)
Step 3: CPU A

$\text{MemA}_i = \text{pri}_{i+1}$

This time $C_{i+1}(H_K(x))$ only contains $H_K(x)$

Are we done?
Not quite...

$C_{i+1}(H_K(x))$
Second Attempt

Step 2': CPU B

Input: $x$

Evaluate relative to $pub_{i+1}$

$C_{i+1}(pri_{i})$

$pri_{i}$

$K$

$H$

Randomize

$C_{i+1}(HK(x))$

$Mem_{B_{i+1}} = C_{i+1}(K)$

$C_{i+1}(K)$
Second Attempt

Step 2': CPU B

Evaluate relative to $pub_{i+1}$

Input: $x$

$C_{i+1}(pri_i)$

$C_{i+1}(pri_i)$

Decrypt

$K$

$H$

$\text{Randomize}$

$\text{Mem}_{B_{i+1}} = C_{i+1}(K)$

$C_{i+1}(H_K(x))$
Second Attempt

Step 2': CPU B

Input: x

Evaluate relative to pub_{i+1}

C_{i+1}(pri_i)

Decrypt

H

Mem_{B_{i+1}} = C_{i+1}(K)

C_{i+1}(K)
Second Attempt

Step 2’: CPU B

Input: $x$

$C_{i+1}(pri_i)$

Evaluate relative to $pub_{i+1}$

$K$

$H$

What did we forget?

$Mem_{B_{i+1}} = C_{i+1}(K)$
Second Attempt

Step 2’: CPU B

Evaluate relative to $pub_{i+1}$

Input: $x$

$C_{i+1}(pri_i)$

$C_{i+1}(K)$ also carries history*

$Mem_{B_{i+1}} = C_{i+1}(K)$

What did we forget?
Third Attempt

Step 2': CPU B

Input: $x$

$C_{i+1}(pri_i)$

Evaluate relative to $pub_{i+1}$

$pri_i$ → Decrypt → $K$ → $H$

$Mem_{B_{i+1}} = C_{i+1}(K)$

$C_{i+1}(K)$
Third Attempt

Step 2”': CPU B

Input: x

$C_{i+1}(pri_i)$

Evaluate relative to $pub_{i+1}$

$MemB_{i+1} = C_{i+1}(K)$

Decrypt

$H$

Randomize

$K$
Third Attempt

Step 2”": CPU B

Evaluate relative to $pub_{i+1}$

Input: $x$

$C_{i+1}(pri_i)$

$Mem_{B_{i+1}} = C_{i+1}(K)$

$C_{i+1}(K)$

$C_{i+1}(pri_i)$

Decrypt

$K$

$H$

Randomize
Third Attempt

Step 2”: CPU B

Evaluate relative to $pub_{i+1}$

Input: $x$

$C_{i+1}(pri_i)$

$Mem_{B_{i+1}} = C_{i+1}(K)$

Now it works!
Complete Construction

Memory A: \( pri_i \)
Randomness: \( r_{gen} \)

\[ pub_{i+1}, pri_{i+1} = \text{KeyGen}(r_{gen}) \]
\[ C_{pri} = \text{Enc}(pri_i, pub_{i+1}) \]
Set Memory A = \( pri_{i+1} \)

\[ Y = \text{Dec}(C'_{reply}; pri_{i+1}) \]
Output \( Y \)

Memory B: \( C_K \)
Randomness: \( r, r' \)

\[ C_{reply} = \text{Evaluate}(C_{pri}, C_K, x, H_K(x); r) \]
\[ C'_K = \text{Evaluate}(C_{pri}, C_K, x, K; r') \]

\[ C'_{reply} = \text{Randomize}(C_{reply}; r) \]
set Memory B = \( \text{Randomize}(C'_K; r') \)
Complete Construction

Memory A: $pri_i$
Randomness: $r_{gen}$

$pub_{i+1}, pri_{i+1} = \text{KeyGen}(r_{gen})$
$C_{pri} = \text{Enc}(pri_i, pub_{i+1})$
Set Memory A = $pri_{i+1}$

$Y = \text{Dec}(C'_{\text{reply}}; pri_{i+1})$
Output $Y$

Memory B: $C_K$
Randomness: $r, r'$

$C_{\text{reply}} = \text{Evaluate}(C_{pri}, C_K, x, H_K(x); r)$
$C'_{K} = \text{Evaluate}(C_{pri}, C_K, x, K; r')$

$C'_{\text{reply}} = \text{Randomize}(C_{\text{reply}}; r)$
set Memory B = $\text{Randomize}(C'_{K}; r')$
Proof
Hybrid 1

Memory A: $\text{pri}_i$
Randomness: $r_{\text{gen}}$

$\text{pub}_{i+1}, \text{pri}_{i+1} = \text{KeyGen}(r_{\text{gen}})$

$C_{\text{pri}} = \text{Enc}(\text{pri}_i, \text{pub}_{i+1})$
Set Memory A = $\text{pri}_{i+1}$

$Y = \text{Dec}(C'_{\text{reply}}; \text{pri}_{i+1})$
Output $Y$

Memory B: $C_K$
Randomness: $r, r'$

$C_{\text{reply}} = \text{Evaluate}(C_{\text{pri}}, C_K, x, H_K(x); r)$

$C'_{K} = \text{Evaluate}(C_{\text{pri}}, C_K, x, K; r')$

$C'_{\text{reply}} = \text{Randomize}(C_{\text{reply}}; r)$
set Memory B = Randomize($C'_{K}; r'$)
Hybrid 1

Memory A: $pri_i$
Randomness: $r_{gen}$

$pub_{i+1}, pri_{i+1} = \text{KeyGen}(r_{gen})$

$C_{pri} = \text{Enc}(pri_i, pub_{i+1})$

Set Memory A = $pri_{i+1}$

$Y = \text{Dec}(C'_{\text{reply}}; pri_{i+1})$

Output $Y$

Memory B: $C_K$
Randomness: $r, r'$

$C_{\text{reply}} = \text{Evaluate}(C_{pri}, C_K, x, H_K(x); r)$

$C'_K = \text{Evaluate}(C_{pri}, C_K, x, K; r')$

$C'_{\text{reply}} = \text{Randomize}(C_{\text{reply}}; r)$

set Memory B = $\text{Randomize}(C'_K; r')$

$C''_{\text{reply}} = \text{Enc}(H_K(x), pub_{i+1})$
Hybrid 1

Memory A: \( pri_i \)
Randomness: \( r_{\text{gen}} \)

\[ \text{pub}_{i+1}, \text{pri}_{i+1} = \text{KeyGen}(r_{\text{gen}}) \]
\[ C_{\text{pri}} = \text{Enc}(\text{pri}_i, \text{pub}_{i+1}) \]
Set Memory A = \( \text{pri}_{i+1} \)

Y = Dec(\( C'_{\text{reply}} \); \( pri_{i+1} \))
Output Y

Memory B: \( C_K \)
Randomness: \( r, r' \)

\[ C_{\text{reply}} = \text{Evaluate}(C_{\text{pri}}, C_K, x, H_K(x); r) \]
\[ C'_K = \text{Evaluate}(C_{\text{pri}}, C_K, x, K; r') \]
\[ C'_{\text{reply}} = \text{Randomize}(C_{\text{reply}}; r) \]
set Memory B = \( \text{Randomize}(C'_K; r') \)

\[ C''_{\text{reply}} = \text{Enc}(H_K(x), \text{pub}_{i+1}) \]

Doesn’t change the distribution
Hybrid 2

Memory A: $pri_i$
Randomness: $r_{gen}$

$pub_{i+1}, pri_{i+1} = \text{KeyGen}(r_{gen})$
$C_{pri} = \text{Enc}(pri_i, pub_{i+1})$
Set Memory A = $pri_{i+1}$

$C_{reply} = \text{Evaluate}(C_{pri}, C_K, x, H_K(x); r)$

$C_{reply} = \text{Randomize}(C_{reply}; r)$
set Memory B = $\text{Randomize}(C_K; r')$

$Y = \text{Dec}(C_{reply}; pri_{i+1})$
Output $Y$

$C''_{reply} =$ $\text{Enc}(H_K(x), pub_{i+1})$
Hybrid 2

Memory A: \( pri_i \)
Randomness: \( r_{gen} \)

\[ \text{pub}_{i+1}, pri_{i+1} = \text{KeyGen}(r_{gen}) \]
\[ C_{pri} = \text{Enc}(pri_i, pub_{i+1}) \]
Set Memory A = \( pri_{i+1} \)

\[ Y = \text{Dec}(C'_{\text{reply}}; pri_{i+1}) \]
Output \( Y \)

Memory B: \( C_K \)
Randomness: \( r, r' \)

\[ C_{\text{reply}} = \text{Evaluate}(C_{pri}, C_K, x, H_K(x); r) \]
\[ C'_K = \text{Evaluate}(C_{pri}, C_K, x, K; r') \]
\[ C'_{\text{reply}} = \text{Randomize}(C_{\text{reply}}; r) \]
set Memory B = \( C''_K \)

\[ C''_{\text{reply}} = \text{Enc}(H_K(x), pub_{i+1}) \]
\[ C''_K = \text{Enc}(0...0, pub_{i+1}) \]
Hybrid 2

Memory A: $pri_i$
Randomness: $r_{gen}$

$pub_{i+1}, pri_{i+1} = KeyGen(r_{gen})$
$C_{pri} = Enc(pri_i, pub_{i+1})$
Set Memory A = $pri_{i+1}$

$Y = Dec(C_{reply} ; pri_{i+1})$
Output Y

Memory B: $C_K$
Randomness: $r, r'$

$C_{reply} = Evaluate(C_{pri}, C_K, x, H_K(x) ; r)$
$C'_K = Evaluate(C_{pri}, C_K, x, K ; r')$
$C'_{reply} = Randomize(C_{reply} ; r)$
set Memory B = $C''_K$

Changes the distribution completely

$C''_{reply} = Enc(H_K(x), pub_{i+1})$
$C''_K = Enc(0...0, pub_{i+1})$
Why Should This Work?

• Very informally: Ciphertexts are incompressible.

• This means that leakage on B can help only if Adv knows enough about pri

• But Adv sees only leakage on pri which is insufficient to break semantic security
Open Questions

• Can we get rid of the leak-free component?
• Granularity of leakage.
Thank you!