Implementing "Practical leakage-resilient symmetric cryptography"

Daniel J. Bernstein

University of Illinois at Chicago,
Technische Universiteit Eindhoven

CHES 2012 paper
“Practical leakage-resilient
symmetric cryptography”
(Faust, Pietrzak, Schipper)
explains how to
“protect against *realistic*
side-channel attacks.”

CHES 2012 paper
"Practical leakage-resilient
symmetric cryptography"
(Faust, Pietrzak, Schipper)
explains how to
"protect against *realistic*
side-channel attacks."

Sounds great!
But is it secure?

CHES 2012 paper
"Practical leakage-resilient
symmetric cryptography"
(Faust, Pietrzak, Schipper)
explains how to
"protect against *realistic*
side-channel attacks."

Sounds great!
But is it secure?

Will an implementor
doing what this paper says
actually end up with a
side-channel-protected cipher?

The TCC view:

“What do you mean?

It’s *provably* secure!

We have proofs and theorems!”

The TCC view:
“What do you mean?
It’s *provably* secure!
We have proofs and theorems!”

Macbeth’s view:
“It is a tale
told by an idiot,
full of sound and fury,
signifying nothing.”

The TCC view:
"What do you mean?
It's *provably* secure!
We have proofs and theorems!"

Macbeth's view:
"It is a tale
told by an idiot,
full of sound and fury,
signifying nothing."

My view: Carefully evaluating
side-channel security
requires an implementation.
$\Rightarrow$ Let's implement the cipher.

Prerequisite: “F”,
a “PRF” (or a “weak PRF”)
mapping a k-bit key
and an ℓ-bit nonce
to a $2k$-bit output.

Prerequisite: "F",
a "PRF" (or a "weak PRF")
mapping a k-bit key
and an ℓ-bit nonce
to a $2k$-bit output.

Hmmm, this is vague.
What's k? ℓ? F?
Practical cryptography
requires complete specification.

Prerequisite: “F”,
a “PRF” (or a “weak PRF”)
mapping a k-bit key
and an ℓ-bit nonce
to a $2k$-bit output.

Hmmm, this is vague.
What’s k? ℓ? F?
Practical cryptography
requires complete specification.

My best guesses:
$k = 128$; $\ell = 127$;
$F_K(p) = \text{AES}_K(0p)\,\text{AES}_K(1p)$.

First-level cipher Γ:

Input: 128-bit key K;
standard random 32639-bit string
$p = (p_0, p_1, \ldots, p_{255}, p_{256})$;
256-bit nonce
$n = (n_0, n_1, \ldots, n_{255})$.

First-level cipher Γ:

Input: 128-bit key K;
standard random 32639-bit string
$p = (p_0, p_1, \ldots, p_{255}, p_{256})$;
256-bit nonce
$n = (n_0, n_1, \ldots, n_{255})$.

Compute
$X_0 = K$,
$X_1 = \text{AES}_{X_0}(n_0 p_0)$,
$X_2 = \text{AES}_{X_1}(n_1 p_1)$, ...,
$X_{256} = \text{AES}_{X_{255}}(n_{255} p_{255})$.

First-level cipher Γ:

Input: 128-bit key K;
standard random 32639-bit string
$p = (p_0, p_1, \ldots, p_{255}, p_{256})$;
256-bit nonce
$n = (n_0, n_1, \ldots, n_{255})$.

Compute
$X_0 = K$,
$X_1 = \text{AES}_{X_0}(n_0 p_0)$,
$X_2 = \text{AES}_{X_1}(n_1 p_1)$, ...,
$X_{256} = \text{AES}_{X_{255}}(n_{255} p_{255})$.

Output: 256-bit string
$\text{AES}_{X_{256}}(p_{256} 0)\,\text{AES}_{X_{256}}(p_{256} 1)$.

The final cipher:

Input:

384-bit key K_0, K_1, K_2;

512-bit plaintext (a_0, b_0).

The final cipher:

Input:

384-bit key K_0, K_1, K_2;

512-bit plaintext (a_0, b_0).

Compute

$(a_1, b_1) = (a_0, b_0 \oplus \Gamma_{K_0}(a_0))$;

$(a_2, b_2) = (a_1 \oplus \Gamma_{K_1}(b_1), b_1)$;

$(a_3, b_3) = (a_2, b_2 \oplus \Gamma_{K_2}(a_2))$.

The final cipher:

Input:
384-bit key K_0, K_1, K_2;
512-bit plaintext (a_0, b_0).

Compute
$(a_1, b_1) = (a_0, b_0 \oplus \Gamma_{K_0}(a_0))$;
$(a_2, b_2) = (a_1 \oplus \Gamma_{K_1}(b_1), b_1)$;
$(a_3, b_3) = (a_2, b_2 \oplus \Gamma_{K_2}(a_2))$.

Output:
512-bit ciphertext (a_3, b_3).

I implemented this cipher during a talk this morning.

I implemented this cipher during a talk this morning.

“Code simplicity?”

I implemented this cipher during a talk this morning.

"Code simplicity?" Not bad, assuming AES is provided.
I used AES from OpenSSL.

I implemented this cipher during a talk this morning.

“Code simplicity?” Not bad, assuming AES is provided.
I used AES from OpenSSL.

“Validation status?”

I implemented this cipher
during a talk this morning.

"Code simplicity?" Not bad,
assuming AES is provided.
I used AES from OpenSSL.

"Validation status?" Bad.
Surely there are bugs.
Practical cryptography
requires test vectors.

I implemented this cipher during a talk this morning.

“Code simplicity?” Not bad, assuming AES is provided. I used AES from OpenSSL.

“Validation status?” Bad. Surely there are bugs. Practical cryptography requires test vectors.

“Source of random p?”

I implemented this cipher during a talk this morning.

"Code simplicity?" Not bad, assuming AES is provided. I used AES from OpenSSL.

"Validation status?" Bad. Surely there are bugs. Practical cryptography requires test vectors.

"Source of random p?" Bad. I used C's `random()`.

I implemented this cipher
during a talk this morning.

"Code simplicity?" Not bad,
assuming AES is provided.
I used AES from OpenSSL.

"Validation status?" Bad.
Surely there are bugs.
Practical cryptography
requires test vectors.

"Source of random p?" Bad.
I used C's `random()`.
I'm going to hell.

“Code availability?”

“Code availability?” Good.

cr.yp.to/aesgonewild.html

“Code availability?” Good.
cr.yp.to/aesgonewild.html

“Speed?”

“Code availability?” Good.
cr.yp.to/aesgonewild.html

“Speed?” Horrifying.
Encrypting 64 bytes:
close to 1 million cycles
on one core of my laptop.

“Code availability?” Good.
cr.yp.to/aesgonewild.html

“Speed?” Horrifying.
Encrypting 64 bytes:
close to 1 million cycles
on one core of my laptop.
But *faster than FHE*.

"Code availability?" Good.

cr.yp.to/aesgonewild.html

"Speed?" Horrifying.

Encrypting 64 bytes:
close to 1 million cycles
on one core of my laptop.
But *faster than FHE*.

"Security?" Unclear!
Try hyperthreading, DPA, etc.
Maybe chosen-n templates
will discover secret n?

Don't let slow ciphers
evade security evaluation.