Crypto Based on Physical Assumptions

Saba Eskandarian
10/27/2015
Outline

● Assumptions in Crypto
● The Physical Connection
● Hardware Assumptions
  ○ Tamper-Proof Hardware (TPH)
  ○ Physically Uncloneable Functions (PUFs)
● Practical Implementations
Something from Nothing: Assumptions in Crypto
Something from Nothing: Assumptions in Crypto
Something from Nothing: Assumptions in Crypto
Something from Nothing: Assumptions in Crypto

Are we building a house of cards?
Assumptions

RAM/Turing Machine Model

Factoring, Discrete Log, ...
Assumptions

RAM/Turing Machine Model

Factoring, Discrete Log, ...

Quantum Model

Many popular assumptions break,

but OWFs seem to exist
Everything is Eventually Physical
Hardware Assumptions: A New Foundation
Tamper-Proof Hardware: Definition

Parties can create and pass back and forth black boxes that compute poly-size circuits:

- Code of the circuit is hidden from user, known to creator
- Two models: Stateful and Stateless TPH
One-Time Programs (OTP): Definition \textsuperscript{[GISVW’10]}

- TPH that self-destructs after a single use
- Special case of stateful TPH
- OTM: OTP that does OT
Tamper-Proof Hardware: Results (Stateful) [GISVW’10]

Do we get secure computation for free?
Tamper-Proof Hardware: Results (Stateful) [GISVW'10]

Do we get secure computation for free?

Not so easy: delayed choice problem
Tamper-Proof Hardware: Results (Stateful) [GISVW’10]

Do we get secure computation for free?

Not so easy: delayed choice problem

Spoiler: There is a way around this and arbitrary secure computation is possible with stateful TPH
Can we get unconditional OT using stateless TPH?
Tamper-Proof Hardware: Results (Stateless) [GIMS’10]

Can we get unconditional OT using stateless TPH?

First Attempt: Sender sends (stateless) OTM
Tamper-Proof Hardware: Results (Stateless) [GIMS’10]

Can we get unconditional OT using stateless TPH?

**First Attempt:** Sender sends (stateless) OTM

Problem: Stateless means the OTM can’t self-destruct

Receiver can query both $s_1$ and $s_2$
Tamper-Proof Hardware: Results (Stateless) [GIMS’10]

Can we get unconditional OT using stateless TPH?

**First Attempt:** Sender sends (stateless) OTM

Problem: Stateless means the OTM can’t self-destruct

Receiver can query both $s_1$ and $s_2$

Cut to the chase: It’s not possible!
Impossibility of (statistical) OT in the Plain Model

Input: \((s_0, s_1), r_A\)

Output: \(s_b\)

\[\text{OT}\]
Impossibility of (statistical) OT in the Plain Model

Input: \((s_0, s_1), r_A\)\hspace{1cm} Input: \(b, r_B\)

\[\text{OT}\]

\[\text{Transcript}\]

\[\text{output: } s_b\]

- \(r_A\) and \(r_B\) are independent given the transcript (views are independent)
- Both parties know everything the other knows about them
- Either Sender or Receiver *always* has an attack
Curious* Adversaries engage in the protocol honestly but ask extra queries:

Indistinguishable from honest execution

Both parties can be curious at once (contrast with 2 malicious parties)

*Not the same as semi-honest or honest-but-curious
Impossibility of (statistical) OT using Stateless TPH

Idea:

1. Curious adversaries ask extra queries from TPH
2. Query responses augment transcript of protocol
3. Both parties now know everything the other knows about them
Quick Aside: Crypto over Physical Inputs
Physically Uncloneable Functions (PUFs)
Physically Uncloneable Functions: Definition

“Random function in a box!”

- unpredictability (noisy?)
- uncloneability

But how to formalize?
OT from Physically Uncloneable Functions [BFSK’11]

Input: \((s_0, s_1)\)

Input: \(b\)
OT from Physically Uncloneable Functions [BFSK’11]

Input: \((s_0, s_1)\)

Input: \(b\)

Create PUF
\[c \in \{0, 1\}^k\]
\[r = \text{PUF}(c)\]
OT from Physically Uncloneable Functions [BFSK’11]

Input: \((s_0, s_1)\)

Input: \(b\)

Create PUF \(c \in \{0,1\}^k\)

\(r = \text{PUF}(c)\)

\(x_0, x_1 \in \{0,1\}^k\)

\(x_0, x_1\)
OT from Physically Uncloneable Functions [BFSK'11]

Input: \((s_0, s_1)\)

Create PUF
\(c \in \{0,1\}^k\)
\(r = \text{PUF}(c)\)

\(x_0, x_1 \in \{0,1\}^k\)

Input: \(b\)

\(v = c \oplus x_b\)
OT from Physically Uncloneable Functions [BFSK’11]

Input: \((s_0, s_1)\)

Create PUF
\(c \in \{0, 1\}^k\)
\(r = \text{PUF}(c)\)

Input: \(b\)

\(x_0, x_1 \in \{0, 1\}^k\)

\(x_0, x_1\)

\(v = c \oplus x_b\)

\(v\)

\(S_0 = s_0 \oplus \text{PUF}(v \oplus x_0)\)

\(S_1 = s_1 \oplus \text{PUF}(v \oplus x_1)\)

\(S_0, S_1\)

\(x_0, x_1\)
OT from Physically Uncloneable Functions [BFSK’11]

Input: \((s_0, s_1)\)

Create PUF

\[ c \in \{0, 1\}^k \]

\[ r = PUF(c) \]

\[ x_0, x_1 \in \{0, 1\}^k \]

\[ v = c \oplus x_b \]

\[ S_0 = s_0 \oplus PUF(v \oplus x_0) \]

\[ S_1 = s_1 \oplus PUF(v \oplus x_1) \]

\[ S_0, S_1 \]

output \( S_b \oplus r \)
Malicious PUFs
Malicious PUFs [OSVW'13]

- Secretly not a PUF at all!
- Compute functionality known to creator
- Stateful/Stateless varieties [DFKLS'14]
(Stateful) Malicious PUFs: OT Impossible! [DFKLS’14]

Define:

$\text{PUF}_S$: PUFs generated by $S$, end in $S$’s hands

$\text{PUF}_R$: PUFs generated by $R$, end in $R$’s hands

$\text{PUF}_R'$: PUFs generated by $R$, end in $S$’s hands

$\text{PUF}_S'$: PUFs generated by $S$, end in $R$’s hands

(We’ll just consider the $\text{PUF}_S$, $\text{PUF}_R$ case because handling $\text{PUF}_S'$, $\text{PUF}_R'$ is more complicated)
(Stateful) Malicious PUFs: OT Impossible! [DFKLS’14]

Attack considering just $\text{PUF}_S, \text{PUF}_R$:

Malicious PUF Behavior:

- Behave as normal PUF
- Save all queries
- Reveal saved queries on secret input
(Stateful) Malicious PUFs: OT Impossible! [DFKLS’14]

Attack considering just $PUF_S$, $PUF_R$:

Malicious PUF Behavior:

- Behave as normal PUF
- Save all queries
- Reveal saved queries on secret input

Now we’re back to the plain model!!
OT from Stateless Malicious PUFs [DFKLS'14]

Input: \((s_0, s_1)\)

Create PUF

\(\text{PUF}_S\)

\[\text{Create PUF}_{R'} \text{PUF}_S\]

\(c \in \{0, 1\}^k\)

\(r = \text{PUF}(c)\)

Input: \(b\)

\(\text{PUF} = \text{PUF}_{R'} \oplus \text{PUF}_S\)

\(x_0, x_1 \in \{0, 1\}^k\)

\(x_0, x_1\)

\(v = c \oplus x_b\)

\(S_0 = s_0 \oplus \text{PUF}(v \oplus x_0)\)

\(S_1 = s_1 \oplus \text{PUF}(v \oplus x_1)\)

\(S_0, S_1\)

Output: \(S_b \oplus r\)
OT from Stateless Malicious PUFs  [DFKLS’14]

Input: $(s_0, s_1)$
Create PUF$_S$

Input: $b$
Create PUF$_R$
$\text{PUF} = \text{PUF}_R \oplus \text{PUF}_S$
$c \in \{0, 1\}^k$
$r = \text{PUF}(c)$

$x_0, x_1 \in \{0, 1\}^k$

$x_0, x_1$ \quad \text{v}

$v = c \oplus x_b$

$S_0 = s_0 \oplus \text{PUF}(v \oplus x_0)$
$S_1 = s_1 \oplus \text{PUF}(v \oplus x_1)$

$S_0, S_1$ \quad \text{output } S_b \oplus r$
Stateless Malicious PUFs: What’s best we can do?

Can we do OT with a single PUF?
Stateless Malicious PUFs: What’s best we can do?

Can we do OT with a single PUF?

Yes! (sender creates it, sends to receiver, and receiver sends it back)
Stateless Malicious PUFs: What’s best we can do?

Can we do OT with a single PUF?

Yes! (sender creates it, sends to receiver, and receiver sends it back)

But not possible if the receiver doesn’t send it back
Physically Uncloneable Functions: Can we build it?
Physically Uncloneable Functions: Can we build it?

PUF implementations proposed from the very start (2001)
Variety of PUF implementations exist
Each has somewhat different properties
Physically Uncloneable Functions: Can we build it?

PUF implementations proposed from the very start (2001)

Variety of PUF implementations exist

Each has somewhat different properties

Takeaway: PUFs exist in real life (caveat coming later)
Some PUFs

- Optical PUFs: the original PUF
- PUFs from IC Manufacturing
  - Silicon, Arbiter, Ring Oscillator PUFs
- Memory-Based PUFs
  - SRAM and Butterfly PUFs
- Coating PUFs
Are the PUFs up to Snuff?

“PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon”

The arbiter PUF responses have a very low entropy and their use in applications with strict unclonability and unpredictability requirements should be carefully considered. Further, the arbiter PUFs are susceptible to changes of their supply voltage and to environmental noise, which significantly increases the bit error rate of the PUF. However, the bit error rate stays within acceptable bounds and can be compensated by existing error correction mechanisms.

The flip-flop and latch PUFs are susceptible to temperature variations, which have a significant effect on the bit error rate and the unpredictability of the PUF responses. Hence, flip-flop and latch PUFs should not be used in an environment, where the adversary can lower the ambient temperature of the PUF, reducing the entropy of the PUF responses.

The SRAM and ring oscillator PUFs achieve almost all desired properties of a PUF: The bit error rate does not change significantly under different operating conditions, the entropy of the PUF responses is high and the responses generated by different PUF instances seem to be independent. However, the ring oscillator exhibits a low min-entropy, which might be problematic in some applications.
Conclusions

- We can do crypto with physical assumptions
- TPH and PUFs as two models with physical assumptions
- Many (but not entirely) positive results in both models
- Physical constructions are pretty good
- Next step: can we get more positive results in related models?
References


D. Dachman-Soled, N. Fleischhacker, J. Katz, A. Lysyanskaya, and D. Schroder, *Feasibility and infeasibility of secure computation with malicious PUFs*.


V. Goyal, Y. Ishai, M. Mahmoody, A. Sahai, *Interactive Locking, Zero-Knowledge PCPs, and Unconditional Cryptography*.

V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, A. Wadia, *Founding Cryptography on Tamper-Proof Hardware Tokens*.


R. Ostrovsky, A. Scafuro, I. Visconti, and A. Wadia, *Universally Composable Secure Computation with (Malicious) Physically Unclonable Functions*.

Pappu, *Physical One-Way Functions*. 