Towards Provably-Secure Masking Compilers

Formal Adventures in the Land of Masking

François Dupressoir

With Gilles Barthe, Sonia Belaïd, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub

masking@projects.easycrypt.info
Real World Crypto?

- "Provably secure":
  - *machine-checked* proofs...
  - ...on implementations, considering low-level attacks;
  - imply formal definitions for security and adversary model;
  - *but* reduction is not tight;
  - *and* we can disagree on the models and security definitions.

- Making security feasible:
  - develop a tool that takes a (restricted) C program, some hints regarding security goals, and produces a protected C-like program or circuit;
  - "sliding scale": security/performance trade-off can be seen as an explicit parameter to the tool.
Differential Power Analysis

Low-level side-channel attacks:

- adversary has access to oracles with some private state;
- adversary can observe power consumption traces produced by executing the oracle;
- adaptively choosing public inputs to the oracle and observing results and leakage, the adversary tries to infer information about the oracle’s private state.

Very effective:

- On an unprotected AES implementation, a single power trace is enough to recover the entire key!
Modelling DPA Adversaries

Noisy Leakage model
- On oracle queries, adversary receives responses and a noisy leakage trace.
- Security is entropy-based.
- (I have) No hope of formalizing it.

$t$-threshold probing model
- Adversary (adaptively or non-adaptively) chooses at most $t$ locations (variables, nodes, wires) in the circuit to probe;
- Security is simulation-based: any set of probes of size at most $t$ can be simulated without the secrets;
- Fairly simple to formalize properly.

DDF14 (EuroCrypt) show that security in the noisy leakage model is implied* by security in the $t$-threshold probing model.
Masking uses secret-sharing schemes to protect implementation against DPA and other side-channel attacks. For example, using an additive secret-sharing scheme:

- A secret $x$ is split into $m$ shares $[x] = (x_0, \ldots, x_{m-1})$ such that the $x_i$s are uniformly distributed and the joint distribution of any $m-1$ of them is independent from $x$.

$$
\begin{align*}
    x_0 & \leftarrow \mathbb{F}_{256} \\
    x_1 & \leftarrow \mathbb{F}_{256} \\
    x_2 & \leftarrow x \oplus x_0 \oplus x_1
\end{align*}
$$

- Splitting secrets into $m$ shares protects computations against adversaries that can set up $m - 1$ probes.
Except when it doesn’t: Secure Multiplication (ISW’03)

function SecMult([a], [b])
    for 0 ≤ i < m do
        for i < j < m do
            $r_{i,j} \leftarrow \mathbb{F}_2$
            $r_{j,i} \leftarrow a_i \odot b_j \oplus (a_j \odot b_i \oplus r_{i,j})$
        end for
    end for
    for 0 ≤ i < m do
        $c_i \leftarrow a_i \odot b_i$
        for 0 ≤ j < m (i ≠ j) do
            $c_i \leftarrow c_i \oplus r_{i,j}$
        end for
    end for
    return [c]

▶ Any set of t probes in SecMult can be simulated using shares $a_i|_I$ and $b_i|_J$, with $|I| \leq 2t$.

▶ This is evidently true for the addition.
Except when it doesn’t: Secure Multiplication (ISW’03)

\[
\text{function SecMult}(\llbracket a \rrbracket, \llbracket b \rrbracket) \\
\text{for } 0 \leq i < m \text{ do} \\
\text{for } i < j < m \text{ do} \\
\quad r_{i,j} \leftarrow \mathbb{F}_2 \\
\quad r_{j,i} \leftarrow a_i \circ b_j \oplus (a_j \circ b_i \oplus r_{i,j}) \\
\text{for } 0 \leq i < m \text{ do} \\
\quad c_i \leftarrow a_i \circ b_i \\
\quad \text{for } 0 \leq j < m \ (i \neq j) \text{ do} \\
\quad \quad c_i \leftarrow c_i \oplus r_{i,j} \\
\text{return } \llbracket c \rrbracket
\]

- Any set of \( t \) probes in \( \text{SecMult} \) can be simulated using shares \( a_i|_I \) and \( b_i|_I \), with \(|I| \leq 2t\).
- This is evidently true for the addition.
- RP’10: Any set of \( t \) probes in \( \text{SecMult} \) can be simulated using shares \( a_i|_I \) and \( b_i|_J \), with \(|I|, |J| \leq t\).
- Secure only if \( \llbracket a \rrbracket \) and \( \llbracket b \rrbracket \) are independent.
The Problem with Composition

- Assume we can prove that any $d < t$ probes in a core gadget can be simulated using at most $d$ shares of each of its inputs. (Simulation)

- Things go smoothly as long as the DFG is a polytree:
  1. split the set of probes on the circuit between core gadgets;
  2. simulate the last gadget (in some topological ordering);
  3. update the set of probes on its parents;
  4. goto 2.

- As soon as you get a DAG, things fall apart:
  - step 3 may push the set of probes on a particular core gadget above the threshold!
  - cryptographers tell us: “You need a refresh.”
  - but that doesn’t give us a (compositional) proof...
But it helps: what property do good refresh gadgets have that other gadgets don’t?

Strong Simulation: every set of $t_i + t_o$ probes on a strongly simulatable gadget that are split between internal ($t_i$) and output ($t_o$) wires can be simulated using at most $t_i$ shares of each of the gadget’s inputs.

If you have two distinct paths between two program points, one of them should go through a strongly simulatable gadget that is not on the other. (Well-Formedness)
Towards a Sound, Compositional, Optimizing, Proof-Producing Masking Compiler

- Machine-checked proof of strong simulation at all orders for the mask refreshing gadget...
- ... and for ISW/RP’s secure multiplication gadget.
- Well-formedness can be enforced with a simple type system.
  - **Bonus:** Type errors mean “a refresh gadget is needed on this particular input bundle”.
- Reuse well-typed sub-circuits as gadgets without re-typing;
- Standard compiler optimization techniques apply:
  - group linear computations as much as possible;
  - “instruction selection” becomes “gadget selection”.
- Given a set of observations, we can produce a simulator for it (the general simulator is just a giant case).
What’s Left?

- Finish implementing and evaluate against hand-crafted optimized implementations;
  - We compile full AES
  - Better loop support $\rightarrow$ Keccak, AES-CBC
  - Support for multiple base structures $\rightarrow$ AES-GCM
  - Masking lookups in public tables with secret indices (C’14)
  - Complex control-flow

- Automatically prove (strong) simulation for more complex gadgets:
  - We already have a tool that proves security in the threshold probing model directly: optimized AES SBox at order 6 can be proved secure in $\sim$5 min.
  - And it can do much more: ask us about it!