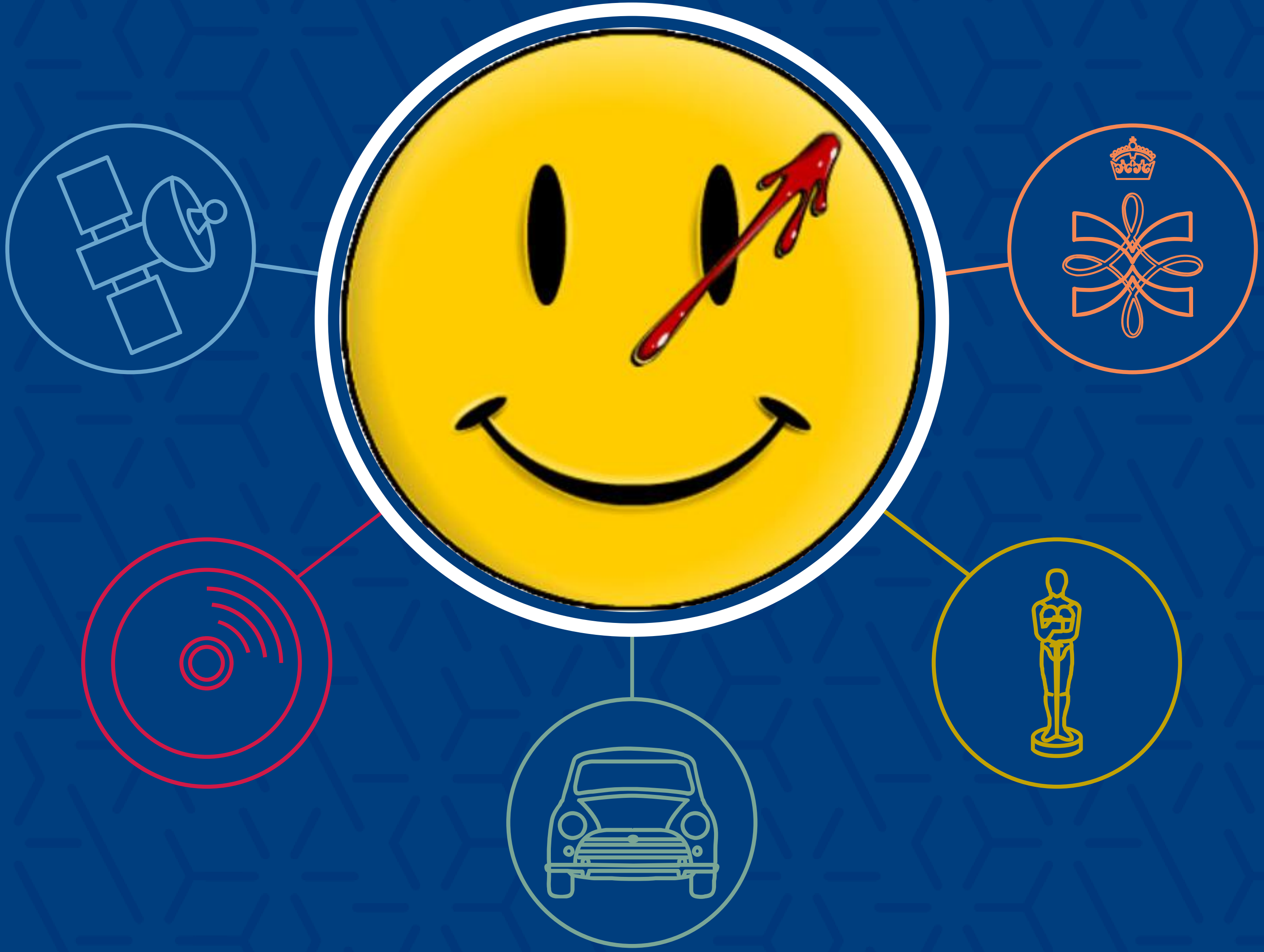
UNIVERSITY OF
SURREY

EasyCrypt: Applying Program Verification Techniques to Cryptography

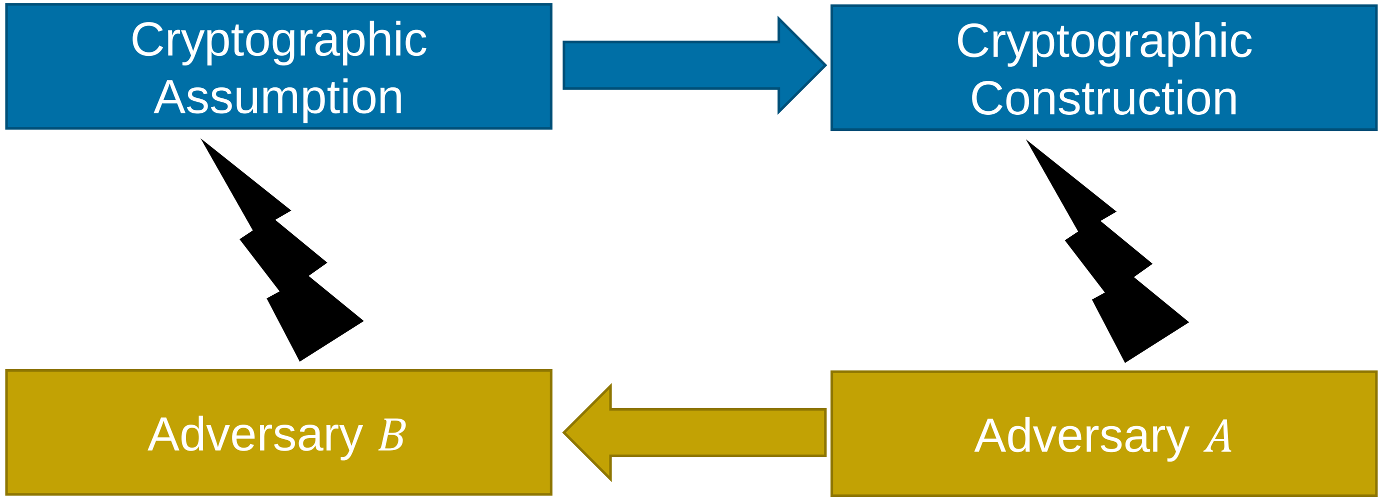
Or where an understanding of concurrency could help

CW-S-REPLS 2019

François Dupressoir
(fdupress@gmail.com)



Security Reductions: A Modern View



From any adversary A against the construction:

- construct an adversary B against the primitive, such that
- if A "breaks the security of" the construction using r_A resources with probability p_A, then B "breaks the security of" the assumption using r_B resources with probability p_B, and
- r_B and p_A are "small" when r_A and p_B are "small"

Tightening Definitions

- Security is *traditionally* modelled using *security games*
  - *Oracles* specify *interfaces* for the adversaries to interact with,
  - A *security experiment* restricts adversary interactions with oracles and defines a *winning condition*,
  - A definition of *adversary advantage* normalizes probability of winning (avoids random chance wins)
- Adversary's resources include time, memory, number of queries to oracles, …

```
experiment $IND\ CPA_E^A$:
  $k \leftarrow_\$ E.keygen();$
  $(m_0, m_1) \leftarrow_\$ A.choose^{E.enc(k,\cdot)}();$
  $b \leftarrow_\$ \{0,1\};$
  $c \leftarrow_\$ E.enc(k, m_b);$
  $b' \leftarrow_\$ A.guess^{E.enc(k,\cdot)}(c);$
  return $b = b'$;
```

$$Adv_E^{INDCPA}(A) = \left|\Pr[IND\ CPA_E^A : \top] - \frac{1}{2}\right|$$

Constructing the inverter: game sequence

```
experiment $Game_0$:
  $(sk, pk) \leftarrow_{\$} P.keygen();$
  $(m_0, m_1) \leftarrow_{\$} A.choose^{H.o}(pk);$
  $b \leftarrow_{\$} \{0,1\};$
  $r \leftarrow_{\$} \{0,1\}^{\kappa};$
  $s \leftarrow P.p(r);$
  $h \leftarrow_{\$} H.o(r);$
  $c \leftarrow s||h \oplus m_b;$
  $b' \leftarrow_{\$} A.guess^{H.o}(c);$
  return $b = b'$;
```

Bypass random

$$\Pr[Game_0 : \top] \leq$$
$$\Pr[Game_1 : \top] + \Pr[Game_1 : r \in H.h]$$

```
experiment $Game_1$:
  $(sk, pk) \leftarrow_{\$} P.keygen();$
  $(m_0, m_1) \leftarrow_{\$} A.choose^{H.o}(pk);$
  $b \leftarrow_{\$} \{0,1\};$
  $r \leftarrow_{\$} \{0,1\}^{\kappa};$
  $s \leftarrow P.p(r);$
  $h \leftarrow_{\$} \{0,1\}^{\kappa'};$
  $c \leftarrow s||h \oplus m_b;$
  $b' \leftarrow_{\$} A.guess^{H.o}(c);$
  return $b = b'$;
```

One-Time Pad

$$\Pr[Game_1 : \top] = \Pr[Game_2 : \top]$$
$$\Pr[Game_1 : r \in H.h] = \Pr[Game_2 : r \in H.h]$$

```
experiment $Game_2$:
  $(sk, pk) \leftarrow_{\$} P.keygen();$
  $(m_0, m_1) \leftarrow_{\$} A.choose^{H.o}(pk);$
  $b \leftarrow_{\$} \{0,1\};$
  $r \leftarrow_{\$} \{0,1\}^{\kappa};$
  $s \leftarrow P.p(r);$
  $h \leftarrow_{\$} \{0,1\}^{\kappa'};$
  $c \leftarrow s||h;$
  $b' \leftarrow_{\$} A.guess^{H.o}(c);$
  return $b = b'$;
```

Security Reductions: A “Post-Modern” View

- EasyCrypt, and CertiCrypt (Barthe et al, POPL 2009) before it, cast the problem of verifying game-based cryptographic proofs as a program verification problem
  - Schemes, oracles, experiments, adversaries are imperative, probabilistic programs (pWhile)
  - pWhile programs are given monadic semantics
  - Claims relating probabilities of events in two programs are reduced to *probabilistic, relational statements about programs*

$$\{P\}c_1 \sim c_2\{Q\} \Leftrightarrow \forall m_1, m_2 . P\ m_1\ m_2 \Rightarrow Q^{\#}\ [\![c_1]\!]_{m_1}\ [\![c_2]\!]_{m_2}$$

where, given a relation Q over memories, $Q^{\#}$ is defined as follows

$$Q^{\#}\ \mu_1\ \mu_2 \Leftrightarrow \exists \mu . \mu_{|m_1} = \mu_1 \wedge \mu_{|m_2} = \mu_2 \wedge \forall (m_1, m_2) \in \mu . Q\ m_1\ m_2$$

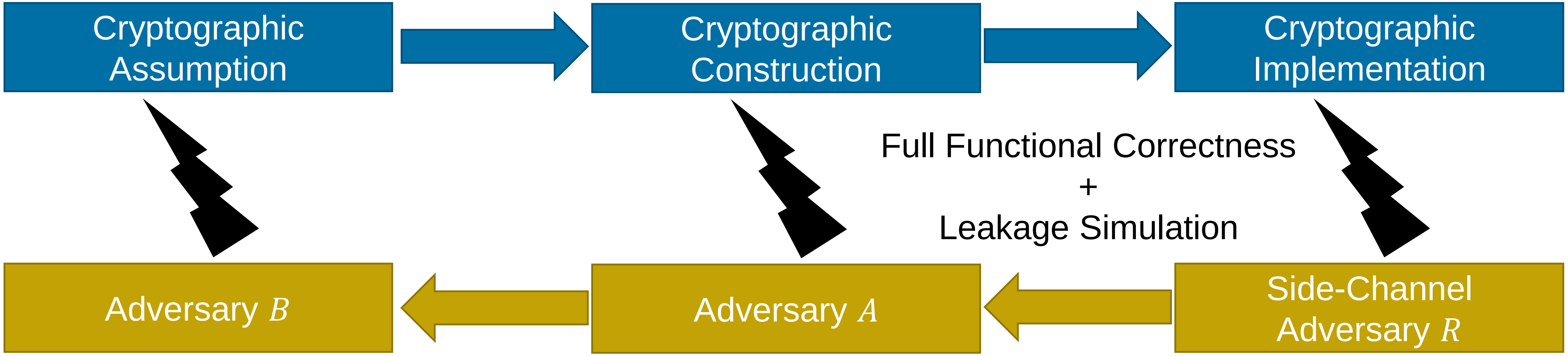
- Proving the lifted relation on final memories consists in constructing a product program that computes joint memory m
  - Done mainly using structural relational Hoare logic,
  - With some trapdoors down to semantics when the programs are too dissimilar.

Achievements

- Standard Cryptographic Primitives
  - OAEP, PSS, CMAC, Merkle-Damgård, SHA-3
  - TLS-MEE-CBC (from TLS1.2)
- Some cryptographic protocols
  - Electronic voting
  - Garbled circuits and Secure Function Evaluation (2-PC)
  - Authenticated Key Exchange

- Applications to cryptographic implementations

Cryptographic Security for Implementations



Cryptographic Assumption

Cryptographic Construction

Cryptographic Implementation

Full Functional Correctness
+
Leakage Simulation

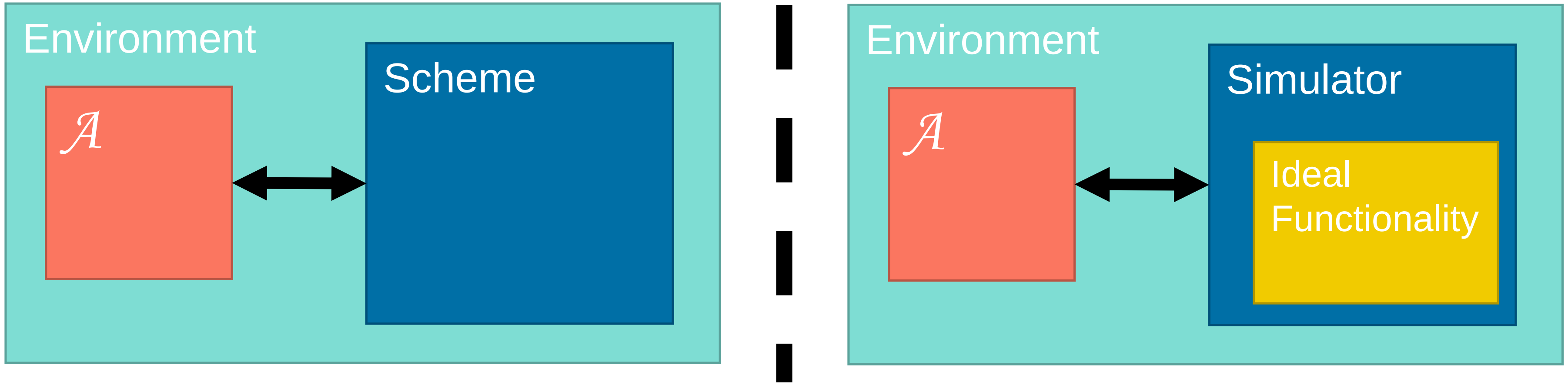
Adversary B

Adversary A

Side-Channel Adversary R

Challenges

- Practice of specifying protocol security moving away from game-based notions
  - Simulation-Based security: no adversary can distinguish between the scheme and a simulator built on top of an ideal functionality (trusted third-party)



  - Composable notions
- As we aim to provide stronger guarantees at lower abstractions, we need finer-grained model of what can go wrong, what leaks

Going Up from the Top

- Interactive systems are increasingly used by the crypto community for compositional security
    - Constructive Cryptography
    - Universal Composability

- The issue is with *interactivity*, not with *composition*
    - Current techniques handle (modular and sequential) composition quite well
    - Issues arise when composition is parallel:

- Having proof tools that support them will be crucial in scaling machine-checked crypto up to larger constructions, and real systems

- Could we leverage ideas from distributed system verification?

Going Down from the Bottom

- Cryptographic implementations are hard to get right
  - Cryptography needs to be fast to be used
  - Getting it to be fast means optimizing
- Non-uniform optimizations may lead to side-channels
  - Execution time
  - Memory accesses (through cache or instruction cache)
  - Power consumption
- Some of these optimizations are done below standard level of reasoning
  - Division on most chips checks for bit size of operands to select long or short division
  - Cache behaviour is hard to reason about
  - Speculative execution, buffered memory …
- We need models of what happens below software to reason about security of software

UNIVERSITY OF
SURREY