RSAConference2019
San Francisco | March 4–8 | Moscone Center

SESSION ID: CRYP-T06

Public Key Encryption Resilient To Post-Challenge Leakage and Tampering Attacks

Suvradip Chakraborty

PhD Graduate Student
Department of Computer Science & Engg.,
IIT Madras, INDIA
@suvradip11

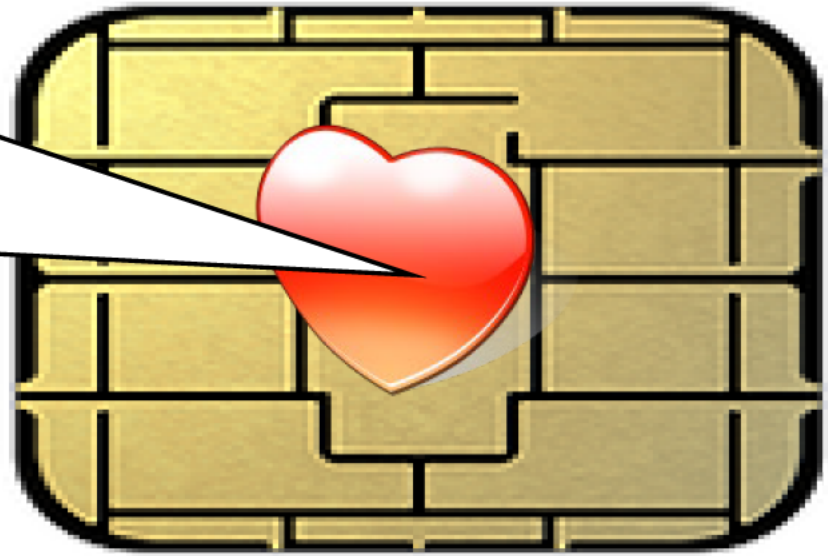
C. Pandu Rangan

Professor
IIT Madras, INDIA

Cryptographic Implementation

very secure

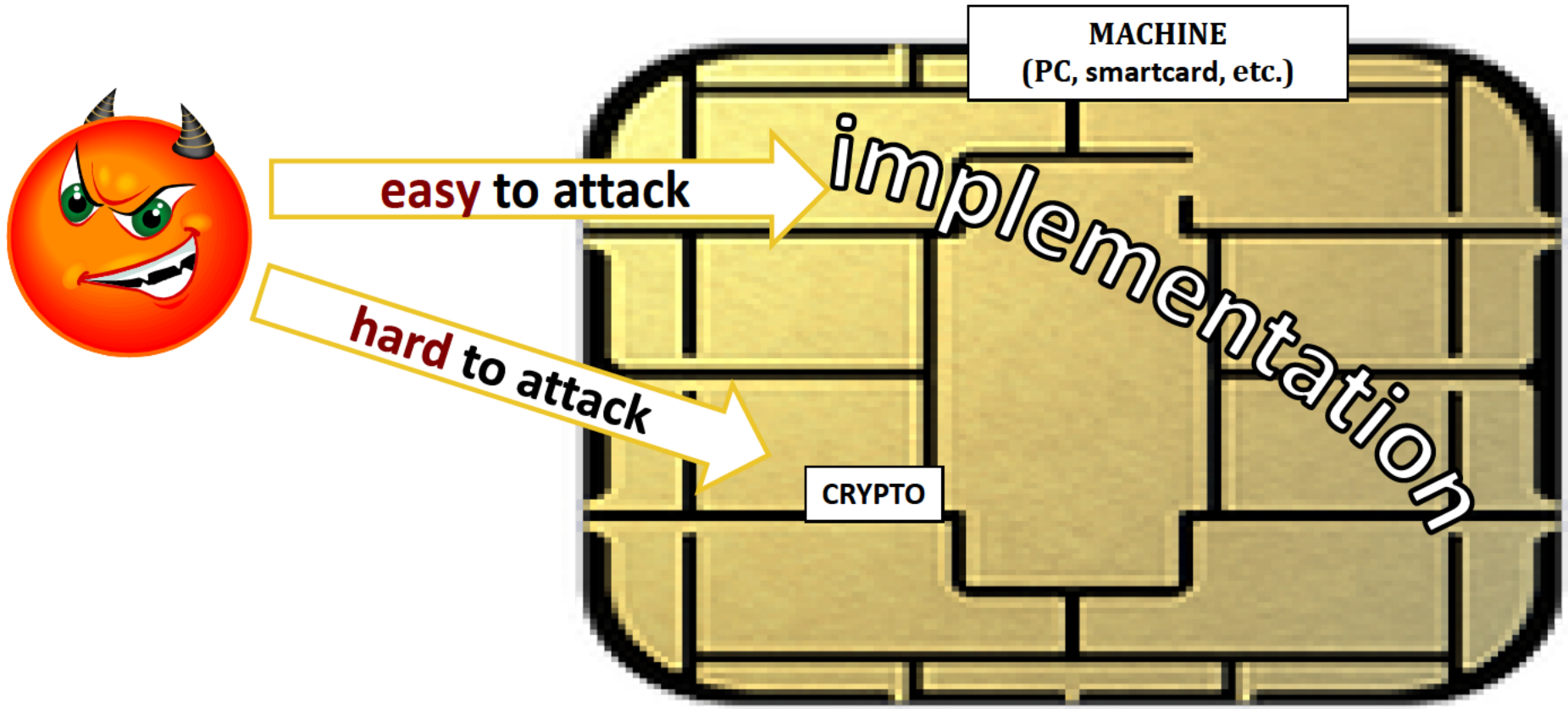
- well-defined mathematical object
- often proof-driven security analysis



much less secure!

side-channel leakage devastating for security

The Problem

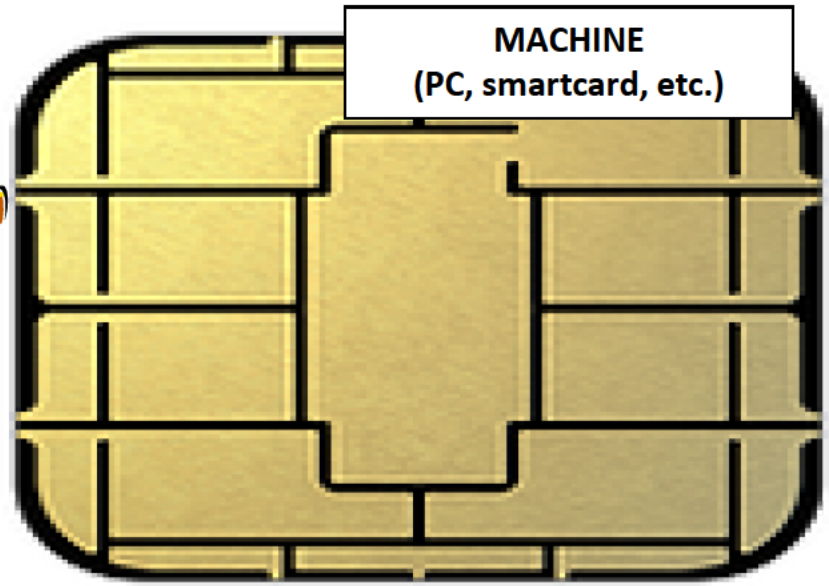


Slide Courtesy: Stefan Dziembowski

Physical attacks

1. Information leakage

(side-channel attacks)

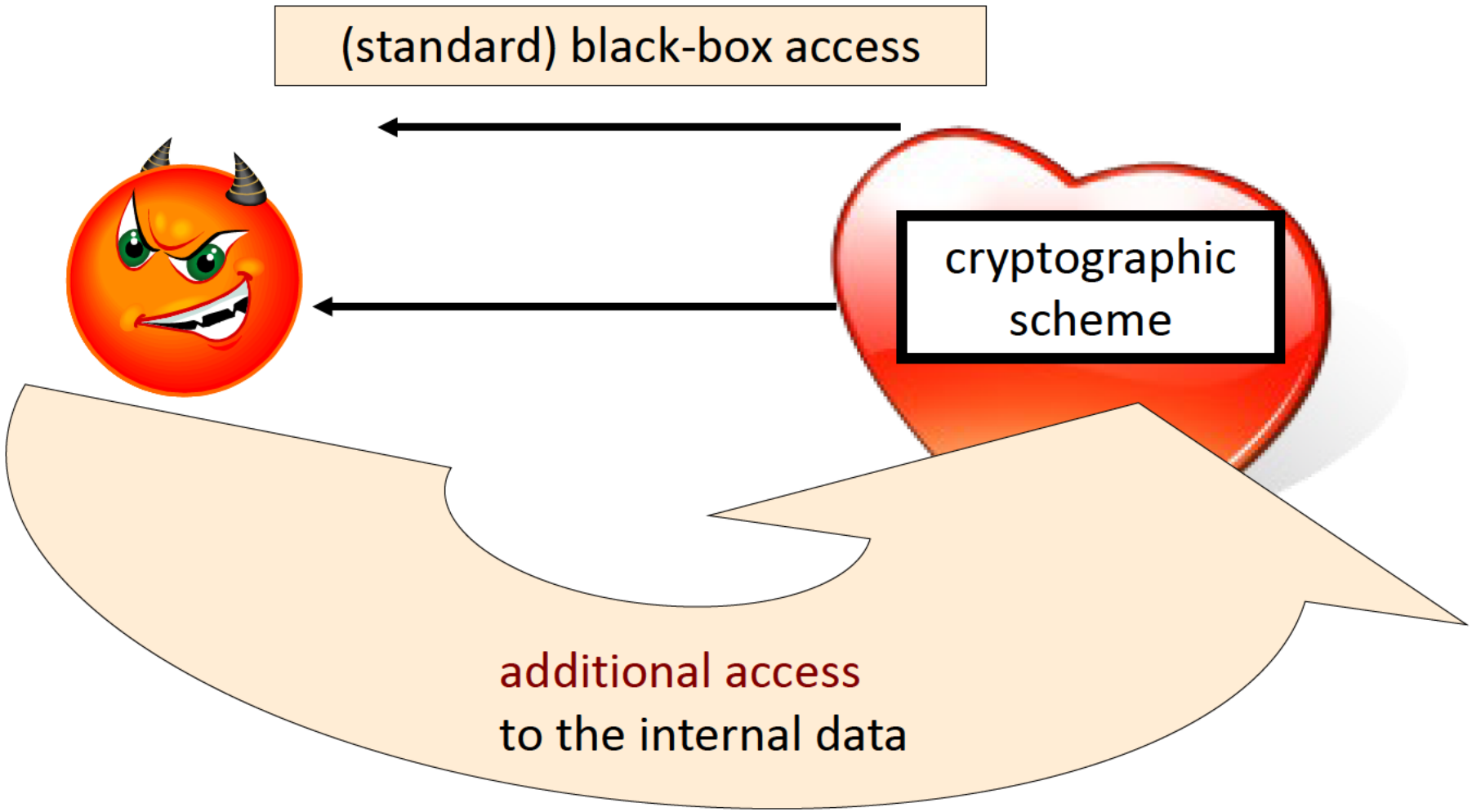


2. Malicious modifications

(tampering attacks)

Slide Courtesy: Stefan Dziembowski

New trend in theory



Slide Courtesy: Stefan Dziembowski

General goal

Come up with attack models that are:

- **realistic** (i.e. they correspond to the real-life adversaries),

- allow to **construct secure schemes**

Bounded Leakage and Tampering (BLT) [DFMV13]

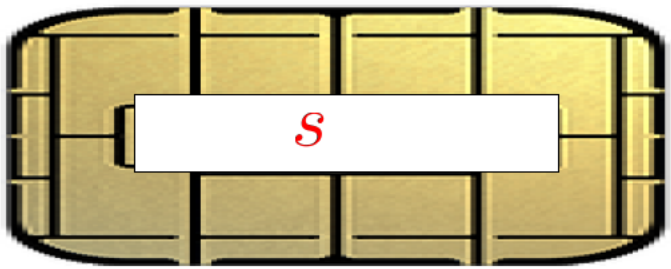
length-shrinking $h(s)$

Input X

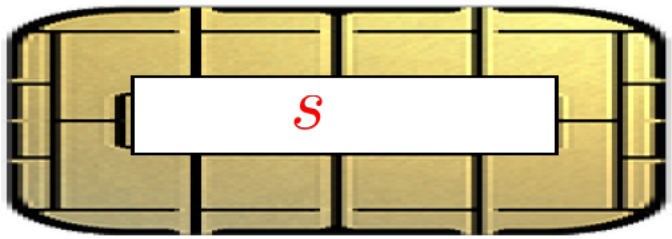
$T_i \in \mathcal{T}$

$G(T_i(s), X)$

Primitive G



Primitive G



"Bounded-leakage model" [D06]
"Memory attacks" [AGV09]

"Bounded-leakage and tampering model" [DFMV13]

Public Key Encryption in the BLT Model

(λ, t)-IND-CCA-BLT-secure PKE

Specified as part of **Setup**

Sample $b \xleftarrow{\$} \{0,1\}$

$params \leftarrow \mathsf{Setup}(\kappa)$

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(params)$

Compute $X_i := \mathcal{L}_i(\mathsf{sk})$

Compute $m_j := \mathsf{Dec}(\mathsf{sk}, C_j)$
$\mathsf{sk} := \mathcal{T}_i(\mathsf{sk})$

Compute $C^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$

Return $m_k := \mathsf{Dec}(\mathsf{sk}, C_k)$

$params$

pk

$\mathcal{L}_i$

X_i

$\mathcal{T}_i, C_j$

m_j

m_0, m_1

C^*

$\mathcal{T}_j = id, C_k \neq C^*$

b'

Subject to the constraint that $\sum_i \mathcal{L}_i(\cdot) \leq \lambda(\kappa)$

Subject to the constraint that # tamp. functions ever queried $\leq t(\kappa)$

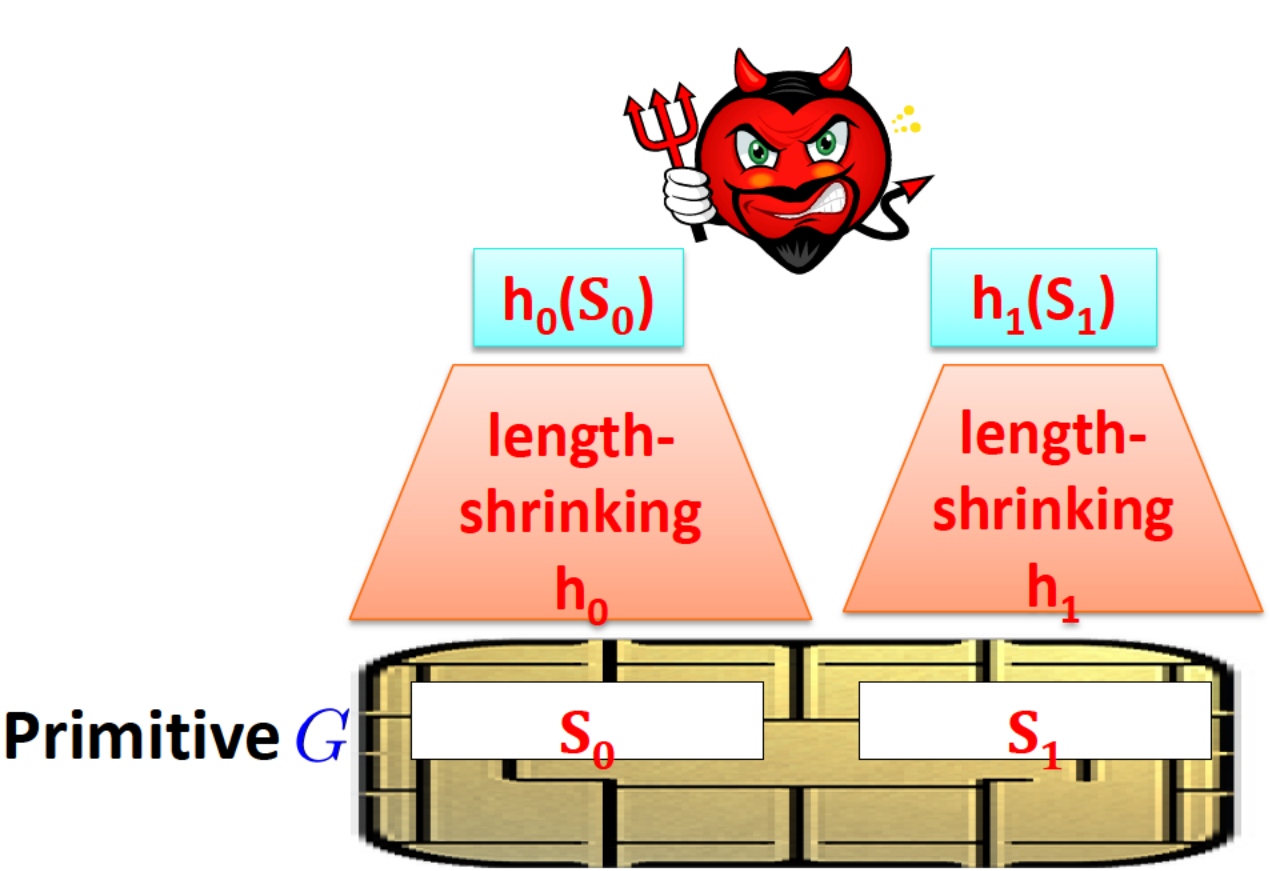
Post Challenge IND-CCA BLT security?

- The prior def. restricts the adversary to **NOT** ask (non-trivial) tampering functions after receiving C^*.
- **Impossible** to tolerate even one post-challenge tampering query [DFMV13]. Similar impossibility holds even if 1 bit of the secret key is allowed to leak after revealing C^*.
- Previous works [DFMV13, FV16] acknowledged that this fact.

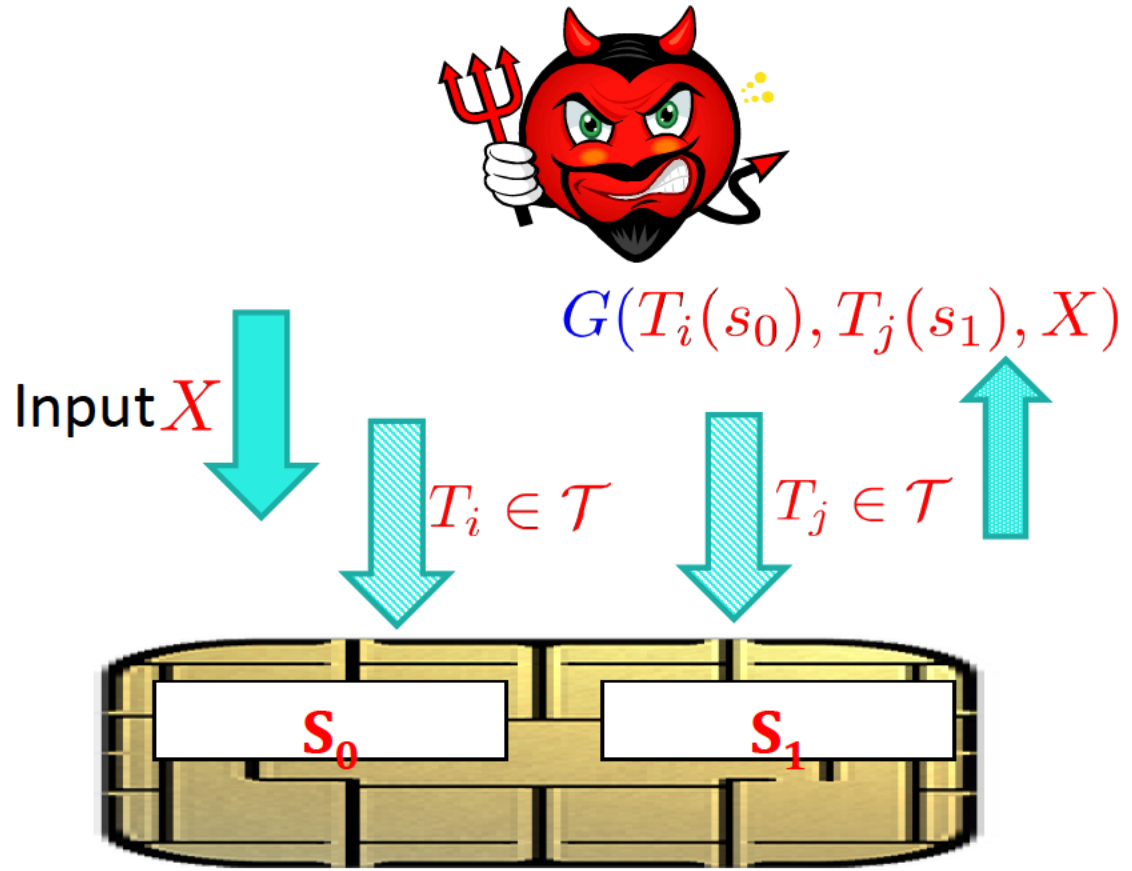
OPEN PROBLEM: Construct *post-challenge* IND-CCA-BLT secure PKE

- **Approach** : Introduce additional (but meaningful) restrictions on the class of allowable leakage & tampering functions (so that they cannot compute the decryption function directly).

Split-State Leakage and Tampering Model



"Split-state model"
[MR04,DP08, HL11…]



Primitive G

IND-CCA-BLT PKE in Split-State model

- Keygen$(\kappa) \rightarrow \big((pk_1, sk_1), (pk_2, sk_2)\big)$ where

$(pk_1, sk_1) \leftarrow \text{Keygen}_1(\kappa)$ & $(pk_2, sk_2) \leftarrow \text{Keygen}_2(\kappa)$. Set $pk = (pk_1, pk_2)$ & $sk = (sk_1, sk_2)$

- Enc$(pk, m) \rightarrow C$ where $pk = (pk_1, pk_2)$ for any message m.
- Dec$(sk = (sk_1, sk_2), C)$: Run $T_1 := \text{Dec}_1(sk_1, C)$ & $T_2 := \text{Dec}_2(sk_2, C)$;

then run $m = \text{Comb}(C, (T_1, T_2))$

IND-CCA-BLT PKE in Split-State model

- The leakage and tampering functions act individually on the secret key components sk_1 and sk_2 respectively.
- The tampering functions should **not** be identity functions w.r.t. C^*.

RSAConference2019

Constructing IND-CCA-BLT PKE in Split State Model

Constructing post-challenge IND-CCA-BLT secure PKE

TWO MAIN STEPS

- Construct entropic ``post-challenge'' IND-CCA-BLT secure PKE---- IND-CCA-BLT secure PKE for ``*entropic*'' messages.

- Compile entropic IND-CCA-BLT security to ``*full-fledged*'' post-challenge IND-CCA-BLT secure PKE scheme.

Entropic PKE in the BLT Model

Intuition:

- Suppose, we start with a message *m** with high min-entropy.
- The adversary obtains i) the public key pk, ii) the challenge ciphertext C^*, (iii) *pre* and *post-challenge* leakage (each bounded by $\lambda(\kappa)$ bits (say)) and access to tampering oracle **both** in pre and post-challenge phase.
- Entropic BLT security guarantees that *m** still ``*looks random*" to the adversary.
  - Formalized by a *real vs. simulation* paradigm
- Validity of the previous impossibility result? **NO**, since the adversary cannot choose messages.

Entropic PKE in the BLT Model--- The Real Game

(λ, t)-ent-IND-CCA-BLT-secure PKE

Specified as part of **Setup**

Sample $m^* \xleftarrow{\$} U_\kappa$

$params \leftarrow \mathsf{Setup}(\kappa)$

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(params)$

Compute $X_i := \mathcal{L}_i(\mathsf{sk})$

Compute $m_j := \mathsf{Dec}(\mathsf{sk}, C_j)$
$\mathsf{sk} := \mathcal{T}_i(\mathsf{sk})$

Compute $C^* \leftarrow \mathsf{Enc}(pk, m^*)$

Return $m_k := \mathsf{Dec}(\mathsf{sk}, C_k)$

$params$

pk

$\mathcal{L}_i$

X_i

$\mathcal{T}_i, C_j$

m_j

C^*

$T_j, C_k \neq C^*$

Subject to the constraint that

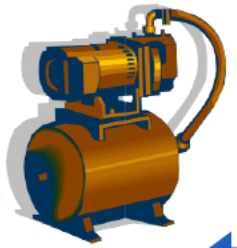
Subject to the constraint that

$\sum_i \mathcal{L}_i(\cdot) \leq \lambda(\kappa)$

tamp. functions ever queried $\leq t(\kappa)$

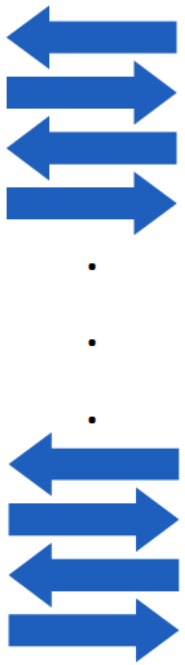
Entropic PKE in the BLT Model--- The Simulated Game

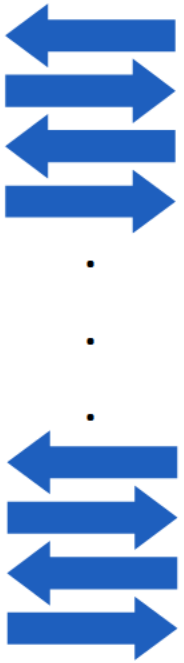
Simulator



m^{sim}

Transcript



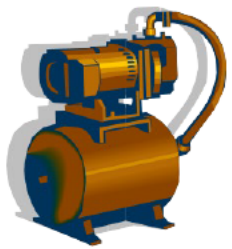


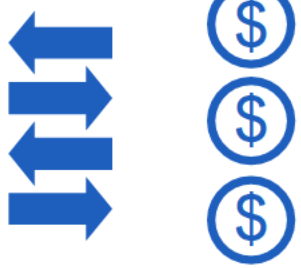
Entropic PKE in the BLT Model--- Defining Security

- ∀ PPT adversary  , ∃ PPT *simulator* S such that

S-views ≈  -views with Challenger

Simulator





Challenger

≈

- The (average) min-entropy of m^{sim} is ``high", even given *S*-views

From Entropic scheme Ω to Full Fledged scheme ∏

Main Ideas:

- The key pairs (pk = (pk_1, pk_2) , sk = (sk_1, sk_2)) of ∏ are generated by running the key-gen of Ω twice *independently*.
- To enc. a message *m* under ∏ do the following:
  - Sample two random strings x_1 and x_2, and run Ω.Enc(pk_1, x_1) and Ω.Enc(pk_2, x_2) to generate C_1 and C_2 resp.
  - Use an (avg.-case) strong randomness extractor EXT(x_1 , x_2) to generate an encapsulation key **K**.
  - Finally, encrypt the message *m* under the key **K** using a (standard) CPA-secure SKE.
- To ensure that the adversary cannot do mix-and-match attack we also need to ``sign" the ciphertexts C_1 and C_2 using a OTS (the verification key of OTS is set as a label).
  - For this, we need to have a labelled entropic post-challenge IND-CCA-BLT scheme.

Security of our resulting scheme ∏

- The entropic scheme Ω guarantees that the strings x_1 and x_2 retains enough entropy even given: 1) C_1 and C_2, 2) pre- and post-challenge leakage and tampering and 3) pk of ∏.
- The split-state model ensures that x_1 and x_2 are ``*independent*''.
- Hence, by the property of EXT, the extracted key **K** is random.
- Finally, the IND-CPA security of SKE ensures that m is ``well-hidden''.
- The OTS ensures that a tag *cannot* be re-used by an adversary in any tampering query.

Instantiating the entropic scheme Ω

- We show that the cryptosystem of Boneh, Hamburg, Halevi and Ostrovsky [**BHHO08**] satisfies our entropic post-challenge IND-CCA-BLT security def.
  - For this, we need to modify the parameters of the BHHO scheme (see our paper for details).
- In slightly more details BHHO cryptosystem satisfies a slighter weaker security definition (which we call entropic ``*restricted*" IND-CCA-BLT security).
- However, the entropic restricted notion can be *upgraded* to entropic notion using generic (simulation-extractable) zero-knowledge proofs.

Putting it Together

- Propose the split-state leakage and tampering model for PKE scheme.
- Propose an entropic version of the above definition.
- Boost the entropic notion of security to full fledged notion of security using OTS, SKE and randomness extractors.
- Instantiate the entropic notion of security using the BHHO cryptosystem appended with appropriate zero-knowledge proof.

Food for Thought and Further Research

- Define and construct continuous leakage and tamper-resilient PKE in the post-challenge setting (without leak and tamper-free hardware assumptions).

- Define an alternate security model for the post-challenge setting (without the split-state restriction)
  - One approach is considering leakage and tampering functions to reside in *low complexity* classes like NC^1 or AC^0.

Full Version of the paper: https://eprint.iacr.org/2018/883