LELEC 2770 - Session 7 - Post-Snowden Cryptography

December 5, 2017

Thanks to the hardware-Trojan resilient architecture exposed during the lecture [3] it is possible to bound the probability of success of an hardware-Trojan adversary. This solution mainly relies on the usage of a semi-honest multi-party computation protocol [1]. In this exercise session, you are asked to build step-by-step an AES Sbox [2] in that hardware-Trojan framework. You will first understand how a secret value can be shared between different parties and how those can compute on it. In the following, M denotes the master circuit, Γ^i_j the mini-circuit i in the sub-circuits j. As a reminder, each sub-circuit is composed of mini-circuits performing a three-party computation. The sub-circuits are independent therefore we may omit the sub-circuit index.

1 Secret sharing

In order to perform the secret sharing, we assume that the mini-circuit Γ^i is able to generate random correlated randomness α_i values. This is defined as $\alpha_1 \oplus \alpha_2 \oplus \alpha_3 = 0$ with two of those uniformly distributed at random. The secret sharing is so performed according to Algorithm 1. You are first asked to deeply understand how this secret sharing works. How can the master M operates to retrieve the shared value v if the shares are held by the mini-circuits ? Once our are done, you can implement the secretsharing and reconstruct functions.

Algorithm 1 Secret sharing and reconstruction.

STEP 1: random generation

- $\Gamma^1, \Gamma^2, \Gamma^3$ runs **GetCorrRandom**
- Γ^1 sends α_1 to M
- Γ^2 sends α_2 to M
- Γ^3 sends α_3 to M

STEP 2: v secret sharing

- M sends $x_3 = v \oplus \alpha_3$ to Γ^1 that holds $\tilde{v}_1 = (\alpha_1, x_3)$
- M sends $x_1 = v \oplus \alpha_1$ to Γ^2 that holds $\tilde{v}_2 = (\alpha_2, x_1)$
- M sends $x_2 = v \oplus \alpha_2$ to Γ^3 that holds $\tilde{v}_3 = (\alpha_3, x_2)$

2 Linear operations

As usual in a multi-party computation, the linear operations do not require any data transfer between the parties (aka mini-circuits). You are asked to derive the way to perform the operations:

- Given three shares $(\tilde{v}_1, \tilde{v}_2, \tilde{v}_3)$ each held by the corresponding mini-circuit, how can those perform the addition with a constant w to retrieve the shares $(\tilde{t}_1, \tilde{t}_2, \tilde{t}_3)$ with $t = v \oplus w$?
- Given three shares $(\tilde{v}_1, \tilde{v}_2, \tilde{v}_3)$ each held by the corresponding mini-circuit, how can those perform the addition with another shared value $(\tilde{w}_1, \tilde{w}_2, \tilde{w}_3)$ to retrieve the shares $(\tilde{t}_1, \tilde{t}_2, \tilde{t}_3)$ with $t = v \oplus w$?

- Given the three shares $(\tilde{v}_1, \tilde{v}_2, \tilde{v}_3)$ each held by the corresponding mini-circuit, how can those perform to square those shares to get $(\tilde{w}_1, \tilde{w}_2, \tilde{w}_3)$ with $w = v^2$.

Once you are done, you can implement the function AddKey and Squaring.

3 Multiplications

The multiplications which are non-linear require some data transfer between the different parties. The multiplications in this three-party computation protocol is performed thanks to Algorithm 2. This example exposes how to obtain the shares of $c = a \odot b$ based on the shares of a and b. How many elements must be sent between two mini-circuits ? Proof the correctness of this algorithm based on [1]. Implement the multiplication function mult

Algorithm 2 Multiplication of secret shared values.

STEP 1: secret sharing

- $\Gamma^1, \Gamma^2, \Gamma^3$ runs **GetCorrRandom** and **SecretShare** on a and b
- Γ^1 holds γ_1, $\tilde{a}_1 = (\alpha_1, x_3)$ and $\tilde{b}_1 = (\beta_1, y_3)$
- Γ^1 holds γ_2, $\tilde{a}_2 = (\alpha_2, x_1)$ and $\tilde{b}_2 = (\beta_2, y_1)$
- Γ^1 holds γ_3, $\tilde{a}_3 = (\alpha_3, x_2)$ and $\tilde{b}_3 = (\beta_3, y_2)$

STEP 2: shared multiplication $c = a \odot b$ 3-out-of-3

- Γ^1 computes $\tilde{c}'_1 = (\alpha_1 \odot \beta_1) \oplus (x_3 \odot y_3) \oplus \gamma_1$
- Γ^2 computes $\tilde{c}'_2 = (\alpha_2 \odot \beta_2) \oplus (x_1 \odot y_1) \oplus \gamma_2$
- Γ^3 computes $\tilde{c}'_3 = (\alpha_3 \odot \beta_3) \oplus (x_2 \odot y_2) \oplus \gamma_3$

STEP 3: 2-out-of-3 from 3-out-of-3 secret sharing

- Γ^1 sends $\tilde{c}'_1$ to Γ^2
- Γ^2 sends $\tilde{c}'_2$ to Γ^3
- Γ^3 sends $\tilde{c}'_3$ to Γ^1
- Γ^1 holds $\tilde{c}_1 = (\tilde{c}'_1 \oplus \tilde{c}'_3, \tilde{c}'_1)$
- Γ^2 holds $\tilde{c}_2 = (\tilde{c}'_2 \oplus \tilde{c}'_1, \tilde{c}'_2)$
- Γ^3 holds $\tilde{c}_3 = (\tilde{c}'_3 \oplus \tilde{c}'_2, \tilde{c}'_3)$

4 AES Sbox

The AES/Rijndael is a blockcipher based on several operations. One of these is called Sbox which consists of an inversion followed by a affine (linear) operation. The inversion of $y \in GF(2^8)$ is obtained by computing $y^{-1} = y^{254}$. This exponentiation is depicted in Figure 1 and is based on squaring and multiplication. You can know easily complete the Sbox function.

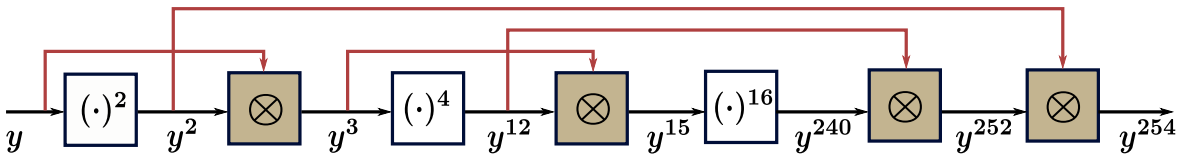

Figure 1: Inversion over $GF(2^8)$

References

[1] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *ACM Conference on Computer and Communications Security*, pages 805–817. ACM, 2016.

[2] J. Daemen and V. Rijmen. Rijndael/aes. In *Encyclopedia of Cryptography and Security*, pages 520–524. Springer, 2005.

[3] S. Dziembowski, S. Faust, and F. Standaert. Private circuits III: hardware trojan-resilience via testing amplification. In *ACM Conference on Computer and Communications Security*, pages 142–153. ACM, 2016.