FHE-MPC Notes

Lecturer: Marcel Keller — Lecture # 9
Scribe: Enrique Larraia — 5th December 2011

In this Lecture we aim to show how to compute AES operations using multiparty computation.

AES Protocol

AES is a block cipher encryption which operates on the field $\mathbb{F}_{2^8}$. The elements are represented modulus the polynomial $G(X) = X^8+X^4+X^3+X+1$, which is irreducible $\mod 2$. That is, we will work with polynomials $a = \sum_{i=0}^{7} a_i X^i \in \mathbb{F}_2[X]/G \simeq \mathbb{F}_{2^8}$. Each element is identified with the byte which corresponds to its coefficient vector $\overline{a} = (a_0, \ldots, a_7)$. The plaintext is given by a 4×4-byte matrix.

AES performs several rounds and within each executes the operations:

SubBytes . An S-Box operation is supplied to every byte of the matrix state. This works as follows: let the affine linear transformation $f_{\mathbf{A},\overline{c}} : \mathbb{F}_{2^8} \to \mathbb{F}2^8 \ : \overline{x} \mapsto \mathbf{A}\overline{x} + \overline{c}$, where $\mathbf{A}$ is a 8×8-matrix with entries in $\mathbb{F}_2$, and $\overline{c} \in \mathbb{F}_{2^8}$ (seen as a 8-length vector with entries in $\mathbb{F}_2$), the S-box is then:

$$\text{S-box}(\overline{a}) = \begin{cases} f_{\mathbf{A},\overline{c}}(\overline{a}^{-1}) & (if)\overline{a} \neq \overline{0}, \\ f_{\mathbf{A},\overline{c}}(\overline{a}) & \text{if } \overline{a} = \overline{0}. \end{cases}$$

ShiftRows . Shift row r_i of the state matrix by a shift of i, for $i = 0, 1, 2, 3$.

MixColumns . The state matrix is multiplied by a fixed, public matrix M with entries in $\mathbb{F}_{2^8}$ (byte entries).

AddRoundKey . Add to the state matrix the round-key matrix.

MPC Setting

We want to compute AES operations sharing inputs among n players. W assume we are using a (linear) shamir secret sharing, so that addition of shares and additions and multiplications by constants can be done locally. Multiplicative operations on shares are achieved by combining the above elementary operations and opening values. We will also need to share random values $r = \sum r_i X^i \in \mathbb{F}_2[X]/G$, but instead of sharing r as an element of the field, we want to share the bits r_i. This is done as follows: every player P_i holds a secret key a_i, which determine certain pseudorandom function f_{a_i} outputting bits. The shared random bit is simply $r_i = \sum_{j=1}^{n} f_{a_j}(a)$, where a is a public value in $\mathbb{F}_{2^8}$. Note that $r_i \in \{0, 1\}$ because $\mathbb{F}_{2^8}$ has characteristic 2.

Sharing Bit Decompositions

Given a shared value $[a] = [\sum a_i X^i]$ sometime AES operates on bytes bitwise, so we need to share the bits of a_i: generate random shared bits $[r_i]$ as explained before. Then locally each player computes the random element r whose bit decomposition is $\sum r_i X^i$, that is, the players compute $[r] = \sum_i^7 [r_i] X^i$ and now set $[c] = [a] + [r] = [a + r]$. Then c is opened to all players and they decompose it into bits. The shared bit decomposition of $[a]$ is

$$[a_i] = c_i + [r_i] = [c + r_i] \qquad \text{for each } i \in \{0, \ldots, 7\},$$

this works since

$$\sum_i^7 a_i X^i = \sum_i^7 (c_i + r_i) X^i = c + r = a + r + r = a.$$

Communication Cost

The operations ShifRows, MixColumns and AddRoundKey come at not cost beacuse they only involve additions, multiplication by constants (public values) and permutations. These are operations that can be done locally.

SubBytes Cost

As we have seen the S-Box is divided into two operations: one inversion and one affine linear transformation. The later again only involve multiplications and additions of constants, so it is done locally. We can now turn to the inversion operation. There are three methods to compute $[a^{-1}]$ from $[a]$.

Square and multiply. Note that for each $a \in \mathbb{F}_{2^8}^*$ we have $a^{-1} = a^{ord(\mathbb{F}_{2^8}^*)-1}$, so the players want to compute $[a^{-1}] = [a^{254}]$ ($\mathbb{F}_{2^8}$ is a field). Since they only know the share of a, they are not sure whether or not $a \in \mathbb{F}_{2^8}^*$ or in other words, whether or not $a \neq 0$. This is not really a problem since $[0^{254}] = [0]$, remember how the S-Box is defined over the byte 0, S-Box(0) $= f_{\mathbf{A},\overline{c}}(0)$, so even if $a = 0$, we get the right S-Box value.

To locally compute a^{254} each player use a variant of the *square and multiply* algorithm with multiplication chains. The chains can be chosen to minimize the number of multiplication or the number of rounds per byte. For example the chain $(1, 2, 4, 8, 9, 18, 36, 55, 72, 127, 254)$ requires 11 multiplications in 9 rounds, which is optimal regarding the number of multiplications. On the other hand the chain $(1, 2, 3, 4, 7, 8, 15, 16, 3132, 63, 54, 127, 254)$ requires 13 multiplications in 8 rounds, which is optimal on the number of rounds.

Masking. We can also exploit the following property: $(ab)^{-1} = a^{-1}b^{-1}$ in $\mathbb{F}_{2^8}$. The idea is mask a with r so it can be opened and therefore its inverse is easily computed (uisng the *extended euclidean algorithm*), then use the property to retrieve the inverse of a.

Choose a random $[r]$, opern $[ar]$ so each player locally computes $(ar)^{-1}$, then $(ar)^{-1}[r] = [a^{-1}r^{-1}r] = [a^{-1}]$. The problem here is that we leak whether or not $a = 0$, since $ar = 0$ for each $r \in \mathbb{F}_{2^8}$ if $a = 0$. To overcome this situation we will find a nonzero value from which $[a^{-1}]$ can be deduced: the players bit-decompose $[a]$ and get $[a_i]$ for $i \in \{0, \ldots, 7\}$, now they compute the control bit

$$[b] = \prod_{i=0}^{7} (1 - [a_i]).$$

Note that if $a = 0$ then $a_i = 0$ for each i, therefore

$$b = \begin{cases} 1 & \text{if } a = 0, \\ 0 & \textit{otherwise}. \end{cases}$$

This means that $a + b \neq 0$ for each $a \in \mathbb{F}_{2^8}$, so the player mask $[a + b]$ with $[r]$ as explained above. The value $[(a + b)r]$ is opened, if is zero, we must have that $r = 0$, so we choose a different r and start again. If $r \neq 0$ then

$$((a + b)r)^{-1}[r] - [b] = [(a + b)^{-1}] - [b] = \begin{cases} [a^{-1}] & \text{if } a \neq 0 (\Leftrightarrow b = 0), \\ [0] & \text{if } a = 0 (\Leftrightarrow b = 1). \end{cases}$$

This method requires 7 multiplications for the control bit, one multiplication for $((a + b)^{-1})[r]$ and one opening.

Offline/Online. This method use the property that if K is a field of characteristic $p > 0$, then $(a + b)^{p^i} = a^{p^i} + b^{p^i}$, this is intuitively clear since in the newton binomial expansion all the coefficients $\left(\begin{matrix} p^i \\ j \end{matrix} \right)$ are a multiple of p (hence 0 in K) except for $j = 0$ and $j = p^i$.

The idea is mask $[a]$ with some random $[r]$, then open $[c] = [a + r]$ and compute $c^2, \ldots, c^{2^7}$. The values $[r^2], \ldots, [r^{2^7}]$ are precomputed in the offline phase, so we can set

$$c^{2^i} + [r^{2^i}] = [(a + r)^{2^i} + r^{2^i}] = [a^{2^i} + 2r^{2^i}] = [a^{2^i}],$$

and hence

$$[a^{254}] = \prod_{i}^{7} [a^{2^i}].$$

The drawback is that the offline phase is not very efficient.

A full explanation of these techniques can be found in the paper *Secure Multiparty AES* of Ivan Damgard and Marcel Keller (eprint 2009/614).