Summer of Science 2022

Topic- Crytography
Midterm Report

Badal Varshney, 19D070015

July 18, 2022

Mentor
Ankit Kumar Misra

Contents

1 Post-Quantum Cryptography

we have equated the notion of efficient adversaries with adversarial algorithms running in (probabilistic) polynomial-time on a classical computer. Thus, when evaluating the security of our schemes we only considered efficient classical attacks. We did not consider the potential impact of quantum computers- that is, computers that rely in an essential way on the principles of quantum mechanics. As we will see,quantum algorithms can in some cases be faster than classical algorithms- possibly much faster- and thus quantum computers can have a dramatic impact on the security of cryptosystems.

1.1 Post-Quantum Symmetric-Key Cryptography

Grover's Algorithm and Symmetric-Key Lengths
To ensure equivalent security against exhaustive-search attacks in the quantum setting, symmetric-key cryptosystems must use keys that are double the length of keys used in the classical setting.

Collision-Finding Algorithms and Hash Functions
To ensure equivalent security against generic collision-finding attacks in the quantum setting, the output length of a hash function must be 50% larger than the output length in the classical setting.

1.2 Shor's Algorithm and its Impact on Cryptography

we have seen quantum algorithms that offer a polynomial speed up as compared to the best classical algorithms for the same problems. These improved algorithms necessitate changes in the underlying parameters of symmetric-key schemes, but do not fundamentally render those schemes insecure. Here,in contrast,we discuss quantum algorithms that result in exponential speed ups for solving certain number-theoretic problems- in particular, we show polynomial-time quantum algorithms for factoring and computing discrete logarithms. The existence of such algorithms means that all the public-key schemes we have seen are insecure (at least asymptotically) against a quantum attacker.

Implications for factoring and computing discrete logarithms
Fix a composite number N that is the product of two distinct primes. Taking any x ϵ Z_N^*, define the function $f_{x,N}$: Z $\longrightarrow$ Z_N^* by

$$f_{x,N}(\mathrm{r}) = [x^r \bmod \mathrm{N}]$$

The key observation is that this function has period ϕ(N) since

$$f_{x,N}(\mathrm{r}+\phi(N)) = [x^{r+\phi(N)} \bmod \mathrm{N}] = [x^r.x^{\phi(N)} \bmod \mathrm{N}] = [x^r \bmod \mathrm{N}]$$

for any r. Thus, for any x ϵ Z_N^* of our choice we can run Shor's algorithm to obtain some period of $f_{x,N}$, i.e., a non zero integer k such that $x^k = 1$ modN. Theorem shows that this enables us to factor N using polynomially many calls to Shor's algorithm and polynomial-time classical computation. (Shor's algorithm in this case runs in time polynomial in the logarithm of the smallest period- which is atmost ϕ(N)- and so this gives a quantum algorithm running in polynomial time overall).

all public-key cryptosystems we have covered thus far can be broken in polynomial time by a quantum computer.

1.3 Post-Quantum Public-Key Encryption

The LWE assumption- Consider the following problem: A matrix Bϵ Z_q^{mxn} is chosen, along with a vector s ϵ Z_q^n. We are then given B and t := [B.s mod q] (i.e.,all operations are done modulo q); the goal is to find any value s' ϵ Z_q^n such that Bs' = t mod q. This problem is easy and can be solved used standard (efficient) linear-algebraic techniques.
Consider next the following variant of the problem- Choose B and s as before, but now also choose a short errorvector e ϵ Z_q^m. (We use the standard Euclidean norm to define the length of vectors. That is, the length of a vector e = $[e1, ..., em]^T$, denoted $||e||$, is simply $\sqrt{\sum_i e_i^2}$. At the moment, we do not quantify what we mean by short.) The value t is now computed as t := [Bs + e mod q] and the goal, given B and t, is to find any s' ϵ Z_q^n such that [t - Bs' mod q] is short. For historical reasons, this is called the learning with errors(LWE) problem. When parameters are chosen appropriately, this problem appears to be significantly more difficult than the previous problem (when there are no errors), and efficient algorithms for solving it- even allowing for quantum algorithms- are not known.

DEFINITION 1: We say the decisional $LWE_{m,q,\psi}$ problem is quantum-hard if for all quantum polynomial-time algorithms **A** there is a negligible function **negl** such that

$$|Pr[\mathbf{B} \longleftarrow Z_q^{mxn}; \mathbf{s} \longleftarrow \psi^n; \mathbf{e} \longleftarrow \psi^m : \mathbf{A(B,[Bs + e}mod q\mathbf{])} = 1] - Pr[\mathbf{B} \longleftarrow Z_q^{mxn}; \mathbf{t} \longleftarrow Z_q^m : \mathbf{A(B,t)} = 1| \leq \text{negl(n)}$$

Clearly,if the decisional $LWE_{m,q,\psi}$ problem is hard then so is the decisional $LWE'_{m,q,\psi}$ problem for any m' $\leq$ m. It is only slightly more difficult to show that increasing the length of s can only make the problem harder.

LWE-Based encryption-

CONSTRUCTION 14.3

Let m, q, ψ be as in the text. Define a public-key encryption scheme as follows:

- Gen: on input 1^n choose uniform $\mathbf{B} \leftarrow \mathbb{Z}_q^{m \times n}$ as well as $\mathbf{s} \leftarrow \psi^n$ and $\mathbf{e} \leftarrow \psi^m$. Set $\mathbf{t} := [\mathbf{B} \cdot \mathbf{s} + \mathbf{e} \bmod q]$. The public key is $\langle \mathbf{B}, \mathbf{t} \rangle$ and the private key is $\mathbf{s}$.
- Enc: on input a public key $pk = \langle \mathbf{B}, \mathbf{t} \rangle$ and a bit b, choose $\hat{\mathbf{s}} \leftarrow \psi^m$ and $\hat{\mathbf{e}} \leftarrow \psi^{n+1}$, and output the ciphertext

$$\mathbf{c}^T := \left[\hat{\mathbf{s}}^T \cdot [\mathbf{B} \mid \mathbf{t}] + \hat{\mathbf{e}}^T + \underbrace{[0, \ldots, 0, b \cdot \lfloor \tfrac{q}{2} \rfloor]}_{n+1} \mod q \right].$$

- Dec: on input a private key $\mathbf{s}$ and a ciphertext $\mathbf{c}^T$, first compute $k := [\mathbf{c}^T \cdot \begin{bmatrix} -\mathbf{s} \\ 1 \end{bmatrix} \bmod q]$. Then output 1 if k is closer to $\lfloor \frac{q}{2} \rfloor$ than to 0 (see text), and 0 otherwise.

CLAIM 14.5 $\left|\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)=1]-\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n)=1]\right|$ *is negligible.*

PROOF The proof is by a direct reduction to the decisional LWE problem as specified in Definition 14.1. Consider the following algorithm D that attempts to solve the decisional $\mathsf{LWE}_{m,q,\psi}$ problem:

Algorithm D:
The algorithm is given $\mathbf{B}\in\mathbb{Z}_q^{m\times n}$ and $\mathbf{t}\in\mathbb{Z}_q^m$ as input.

- Set $pk:=\langle\mathbf{B},\mathbf{t}\rangle$ and run $\mathcal{A}(pk)$ to obtain $m_0,m_1\in\{0,1\}$.
- Choose a uniform bit b, and set

$$\mathbf{c}^T:=\left[\hat{\mathbf{s}}^T\cdot[\mathbf{B}\mid\mathbf{t}]+\hat{\mathbf{e}}^T+[0,\ldots,0,m_b\cdot\lfloor\frac{q}{2}\rfloor]\mod q\right].$$

- Give the ciphertext $\mathbf{c}^T$ to $\mathcal{A}$ and obtain an output bit b'. If $b'=b$, output 1; otherwise, output 0.

Note that D is a quantum polynomial-time algorithm since $\mathcal{A}$ is.
It is immediate that

$$\Pr[\mathbf{B}\leftarrow\mathbb{Z}_q^{m\times n};\mathbf{s}\leftarrow\psi^n;\mathbf{e}\leftarrow\psi^m:D\left(\mathbf{B},[\mathbf{B}\mathbf{s}+\mathbf{e}\bmod q]\right)=1]$$
$$=\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)=1]$$

and

$$\Pr[\mathbf{B}\leftarrow\mathbb{Z}_q^{m\times n};\mathbf{t}\leftarrow\mathbb{Z}_q^m:D(\mathbf{B},\mathbf{t})=1]=\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n)=1].$$

Quantum hardness of the $\mathsf{LWE}_{m,q,\psi}$ problem implies the claim. ■

1.4 Post-Quantum Signatures

Lamport's Signature Scheme-
Let = (Gen, Sign, Vrfy) be a signature scheme, and consider the following experiment for an adversary **A** and parameter n: **The one-time signature experiment** $Sig-forge_{A,\pi}^{1-time}$ (n):

1. Gen(1^n) is run to obtain keys (pk,sk).
2. Adversary **A** is given pk and asks a single query m' to its oracle $Sign_{sk}(.)$. **A** then outputs (m, σ) with m $\neq$ m'.
3. The output of the experiment is defined to be 1 if and only if $Vrfy_{pk}$(m, σ) = 1.

DEFINITION 2 Signature scheme II = (Gen, Sign, Vrfy) is existentially unforgeable under a single-message attack, or is a one-time signature scheme, if for all probabilistic polynomial-time adversaries **A**, there exists a negligible function negl such that:

$$\Pr[\text{Sig-}forge_{A,\pi}^{1-time}(\text{n})=1]\leq\text{negl(n)}.$$

THEOREM: Let l be any polynomial. If H is a one-way function, then Construction 14.9 is a one-time signature scheme.

Proof-

CONSTRUCTION 14.9

Let $H : \{0,1\}^* \to \{0,1\}^*$ be a function. Construct a signature scheme for messages of length $\ell = \ell(n)$ as follows:

- Gen: on input 1^n, proceed as follows for $i \in \{1, \ldots, \ell\}$:
  1. Choose uniform $x_{i,0}, x_{i,1} \in \{0,1\}^n$.
  2. Compute $y_{i,0} := H(x_{i,0})$ and $y_{i,1} := H(x_{i,1})$.

  The public key pk and the private key sk are

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix} \quad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{\ell,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{\ell,1} \end{pmatrix}.$$

- Sign: on input a private key sk as above and a message $m \in \{0,1\}^\ell$ with $m = m_1 \cdots m_\ell$, output the signature $(x_{1,m_1}, \ldots, x_{\ell,m_\ell})$.
- Vrfy: on input a public key pk as above, a message $m \in \{0,1\}^\ell$ with $m = m_1 \cdots m_\ell$, and a signature $\sigma = (x_1, \ldots, x_\ell)$, output 1 if and only if $H(x_i) = y_{i,m_i}$ for all $1 \le i \le \ell$.

The Lamport signature scheme.

Consider the following PPT algorithm $\mathcal{I}$ attempting to invert H:

Algorithm $\mathcal{I}$:
The algorithm is given 1^n and y as input.

1. Choose uniform $i^* \in \{1, \ldots, \ell\}$ and $b^* \in \{0,1\}$. Set $y_{i^*,b^*} := y$.
2. For all $i \in \{1, \ldots, \ell\}$ and $b \in \{0,1\}$ with $(i,b) \neq (i^*, b^*)$:
   - Choose uniform $x_{i,b} \in \{0,1\}^n$ and set $y_{i,b} := H(x_{i,b})$.
3. Run $\mathcal{A}$ on input $pk := \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix}$.
4. When $\mathcal{A}$ requests a signature on the message m':
   - If $m'_{i^*} = b^*$, then $\mathcal{I}$ aborts the execution.
   - Otherwise, $\mathcal{I}$ returns the signature $\sigma = (x_{1,m'_1}, \ldots, x_{\ell,m'_\ell})$.
5. When $\mathcal{A}$ outputs (m, σ) with $\sigma = (x_1, \ldots, x_\ell)$:
   - If $\mathcal{A}$ outputs a forgery at (i^*, b^*), then output x_{i^*}.

Chain-Based Signatures

DEFINITION 14.12 *A* stateful signature scheme *is a tuple of probabilistic polynomial-time algorithms* (Gen, Sign, Vrfy) *satisfying the following:*

1. *The* key-generation algorithm Gen *takes as input a security parameter* 1^n *and outputs* (pk, sk, s_0). *These are called the* public key, private key, *and* initial state, *respectively. We assume pk and sk each has length at least n, and that n can be determined from pk, sk.*

2. *The* signing algorithm Sign *takes as input a private key sk, a value* s_{i-1}, *and a message* $m \in \{0,1\}^*$. *It outputs a signature* σ *and a value* s_i.

3. *The deterministic* verification algorithm Vrfy *takes as input a public key pk, a message m, and a signature* σ. *It outputs a bit b.*

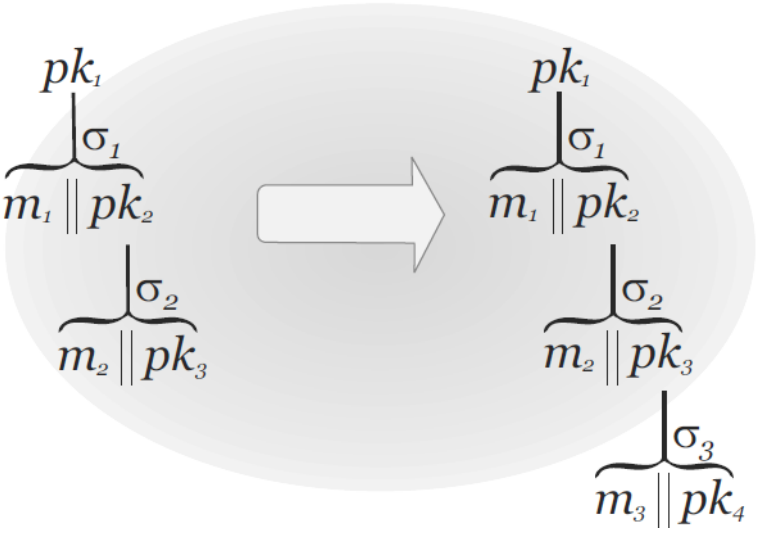


Figure 1: Chain-based signatures: the situation be fore and after signing the third message m3

Tree-Based Signatures

THEOREM 14.15 *Let Π be a one-time signature scheme. Then Construction 14.14 is a secure signature scheme.*

PROOF Let Π^* denote Construction 14.14. Let $\mathcal{A}^*$ be a probabilistic polynomial time adversary, let $\ell^* = \ell^*(n)$ be a (polynomial) upper bound on the number of signing queries made by $\mathcal{A}^*$, and set $\ell = \ell(n) \stackrel{\text{def}}{=} 2n\ell^*(n) + 1$. Note that ℓ upper bounds the number of public keys from Π that are needed to generate ℓ^* signatures using Π^*. This is because each signature in Π^* requires at most $2n$ new keys from Π (in the worst case), and one additional key from Π is used as the actual public key pk_ε.

CONSTRUCTION 14.14

Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme. For a binary string m, let $m|_i \stackrel{\text{def}}{=} m_1 \cdots m_i$ denote the i-bit prefix of m (with $m|_0 \stackrel{\text{def}}{=} \varepsilon$, the empty string). Construct the scheme $\Pi^* = (\mathsf{Gen}^*, \mathsf{Sign}^*, \mathsf{Vrfy}^*)$ as follows:

- Gen^*: on input 1^n, compute $(pk_\varepsilon, sk_\varepsilon) \leftarrow \mathsf{Gen}(1^n)$ and output the public key pk_ε. The private key and initial state are sk_ε.
- Sign^*: on input a message $m \in \{0,1\}^n$, carry out the following.
  1. For $i = 0$ to $n-1$:
     - If $pk_{m|_i 0}, pk_{m|_i 1}$, and $\sigma_{m|_i}$ are not in the state, compute $(pk_{m|_i 0}, sk_{m|_i 0}) \leftarrow \mathsf{Gen}(1^n)$, $(pk_{m|_i 1}, sk_{m|_i 1}) \leftarrow \mathsf{Gen}(1^n)$, and $\sigma_{m|_i} \leftarrow \mathsf{Sign}_{sk_{m|_i}}(pk_{m|_i 0} \,\|\, pk_{m|_i 1})$. In addition, add all of these values to the state.
  2. If σ_m is not yet included in the state, compute $\sigma_m \leftarrow \mathsf{Sign}_{sk_m}(m)$ and store it as part of the state.
  3. Output the signature $\left(\{\sigma_{m|_i}, pk_{m|_i 0}, pk_{m|_i 1}\}_{i=0}^{n-1}, \sigma_m\right)$.
- Vrfy^*: on input a public key pk_ε, message m, and signature $\left(\{\sigma_{m|_i}, pk_{m|_i 0}, pk_{m|_i 1}\}_{i=0}^{n-1}, \sigma_m\right)$, output 1 if and only if:
  1. $\mathsf{Vrfy}_{pk_{m|_i}}(pk_{m|_i 0} \,\|\, pk_{m|_i 1}, \sigma_{m|_i}) \stackrel{?}{=} 1$ for all $i \in \{0, \ldots, n-1\}$.
  2. $\mathsf{Vrfy}_{pk_m}(m, \sigma_m) \stackrel{?}{=} 1$.

A "tree-based" signature scheme.

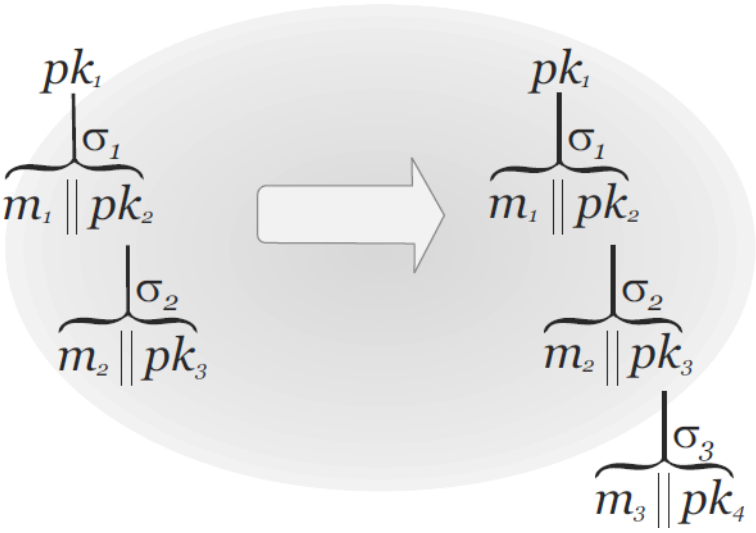


Figure 2: Tree-based signatures

2 Advanced Topics in Public-Key Encryption

We begin with a treatment of trapdoor permutations, a generalization of one-way permutations, and show how to use them to construct public-key encryption schemes.Trapdoor permutations neatly encapsulate the key characteristics of the RSA permutation that make it so useful. As such,they often provide a useful abstraction for designing new cryptosystems.

2.1 Encryption from Trapdoor Permutations

We saw how to construct a CPA-secure public-key encryption scheme based on the RSA assumption. By distilling those properties of RSA that are used in the construction, and defining an abstract notion that encapsulates those properties, we obtain a general template for constructing secure encryption schemes based on any primitive satisfying the same set of properties. Trapdoor permutations turn out to be the "right" abstraction here.

Trapdoor Permutations

As shorthand, we drop explicitmention of Samp and simply refer to trapdoor permutation(Gen, f, Inv). For(I, td) output by Gen we write x $\longleftarrow$ D_I to denote uniform selection of x ϵ D_I (with the understanding that this is done by algorithm Samp).
The second condition above implies that f_I cannot be efficiently inverted without td, but the final condition means that f_I can be efficiently inverted with td. It is immediate that can be modified to give a family of trapdoor permutations if the RSA problem is hard relative to GenRSA, and so we refer to that construction as the RSA trapdoor permutation.

DEFINITION 15.1 *A tuple of polynomial-time algorithms* (Gen, Samp, f, Inv) *is a* family of trapdoor permutations (*or a* trapdoor permutation) *if:*

- *The probabilistic* parameter-generation algorithm Gen, *on input* 1^n, *outputs* (I, td) *with* $|I| \geq n$. *Each value of* I *defines a set* $\mathcal{D}_I$ *that constitutes the domain and range of a permutation (i.e., bijection)* $f_I : \mathcal{D}_I \to \mathcal{D}_I$.
- *Let* Gen_1 *denote the algorithm that results by running* Gen *and outputting only* I. *Then* $(\mathsf{Gen}_1, \mathsf{Samp}, f)$ *is a family of one-way permutations.*
- *Let* (I, td) *be an output of* $\mathsf{Gen}(1^n)$. *The deterministic* inverting algorithm Inv, *on input* td *and* $y \in \mathcal{D}_I$, *outputs* $x \in \mathcal{D}_I$. *We denote this by* $x := \mathsf{Inv}_{\mathsf{td}}(y)$. *It is required that with all but negligible probability over* (I, td) *output by* $\mathsf{Gen}(1^n)$ *and uniform choice of* $x \in \mathcal{D}_I$, *we have*

$$\mathsf{Inv}_{\mathsf{td}}(f_I(x)) = x.$$

Public-Key Encryption from Trapdoor Permutations

DEFINITION 15.2 *Let* $\Pi = (\mathsf{Gen}, f, \mathsf{Inv})$ *be a family of trapdoor permutations, and let* hc *be a deterministic polynomial-time algorithm that, on input* I *and* $x \in \mathcal{D}_I$, *outputs a single bit* $\mathsf{hc}_I(x)$. *We say that* hc *is a* hard-core predicate of Π *if for every probabilistic polynomial-time algorithm* $\mathcal{A}$ *there is a negligible function* negl *such that*

$$\Pr[\mathcal{A}(I, f_I(x)) = \mathsf{hc}_I(x)] \leq \frac{1}{2} + \mathsf{negl}(n),$$

where the probability is taken over the experiment in which $\mathsf{Gen}(1^n)$ *is run to generate* (I, td) *and then* x *is chosen uniformly from* $\mathcal{D}_I$.

THEOREM If π is a family of trapdoor permutations with hard-core predicate hc, then Construction15.4 is CPA-secure.

CONSTRUCTION 15.4

Let $\widehat{\Pi} = (\widehat{\mathsf{Gen}}, f, \mathsf{Inv})$ be a family of trapdoor permutations with hard-core predicate hc. Define a public-key encryption scheme as follows:

- Gen: on input 1^n, run $\widehat{\mathsf{Gen}}(1^n)$ to obtain (I, td). Output the public key I and the private key td.
- Enc: on input a public key I and a message $m \in \{0,1\}$, choose a uniform $r \in \mathcal{D}_I$ subject to the constraint that $\mathsf{hc}_I(r) = m$. Output the ciphertext $c := f_I(r)$.
- Dec: on input a private key td and a ciphertext c, compute the value $r := \mathsf{Inv}_{\mathsf{td}}(c)$ and output the message $\mathsf{hc}_I(r)$.

Public-key encryption from any family of trapdoor permutations.

2.2 The Paillier Encryption Scheme

The Paillier encryption scheme utilizes the group $Z_{N^2}^*$, the multiplicative group of elements in the range 1,...N^2 that are relatively prime to N, for N a product of two distinct primes. To understand the scheme it is helpful to first understand the structure of $Z_{N^2}^*$. A useful characterization of this group is given by the following proposition, which says,among other things, that $Z_{N^2}^*$ is isomorphic to Z_N x Z_N^* for N of the form we will be interested in.

PROPOSITION 15.6 *Let $N = pq$, where p, q are distinct odd primes of equal length. Then:*

1. $\gcd(N, \phi(N)) = 1$.
2. *For any integer $a \geq 0$, we have $(1+N)^a = (1+aN) \bmod N^2$.*

   As a consequence, the order of $(1+N)$ in $\mathbb{Z}_{N^2}^*$ is N. That is, $(1+N)^N = 1 \bmod N^2$ and $(1+N)^a \neq 1 \bmod N^2$ for any $1 \leq a < N$.
3. *$\mathbb{Z}_N \times \mathbb{Z}_N^*$ is isomorphic to $\mathbb{Z}_{N^2}^*$, with isomorphism $f : \mathbb{Z}_N \times \mathbb{Z}_N^* \to \mathbb{Z}_{N^2}^*$ given by*

$$f(a,b) = \left[(1+N)^a \cdot b^N \bmod N^2\right].$$

The Paillier Encryption Scheme

DEFINITION 15.10 *The* decisional composite residuosity problem is hard relative to GenModulus *if for all probabilistic polynomial-time algorithms D there is a negligible function* negl *such that*

$$\left|\Pr[D(N,[r^N \bmod N^2])=1]-\Pr[D(N,r)=1]\right| \le \mathsf{negl}(n),$$

where in each case the probabilities are taken over the experiment in which $\mathsf{GenModulus}(1^n)$ *outputs* (N,p,q)*, and then a uniform* $r \in \mathbb{Z}_{N^2}^*$ *is chosen. (Recall that* $[r^N \bmod N^2]$ *is a uniform element of* $\mathsf{Res}(N^2)$*.)*

The *decisional composite residuosity* (*DCR*) *assumption* is the assumption that there is a GenModulus relative to which the decisional composite residuosity problem is hard.

Encryption. We have described encryption above as though it is taking place in $\mathbb{Z}_N \times \mathbb{Z}_N^*$. In fact it takes place in the isomorphic group $\mathbb{Z}_{N^2}^*$. That is, the sender generates a ciphertext $c \in \mathbb{Z}_{N^2}^*$ by choosing a uniform[1] $r \in \mathbb{Z}_N^*$ and then computing

$$c := [(1+N)^m \cdot r^N \bmod N^2].$$

Observe that

$$c = \left((1+N)^m \cdot 1^N\right) \cdot \left((1+N)^0 \cdot r^N\right) \bmod N^2 \leftrightarrow (m,1)\cdot(0,r),$$

and so $c \leftrightarrow (m,r)$ as desired.

Decryption. We now describe how decryption can be performed efficiently given the factorization of N. For c constructed as above, we claim that m is recovered by the following steps:

- Set $\hat{c} := [c^{\phi(N)} \bmod N^2]$.
- Set $\hat{m} := (\hat{c}-1)/N$. (Note that this is carried out over the integers.)
- Set $m := \left[\hat{m} \cdot \phi(N)^{-1} \bmod N\right]$.

To see why this works, let $c \leftrightarrow (m, r)$ for an arbitrary $r \in \mathbb{Z}_N^*$. Then

$$\begin{aligned}\hat{c} &\stackrel{\text{def}}{=} [c^{\phi(N)} \bmod N^2] \\ &\leftrightarrow (m,\, r)^{\phi(N)} \\ &= \Big([m \cdot \phi(N) \bmod N],\ [r^{\phi(N)} \bmod N]\Big) \\ &= \big([m \cdot \phi(N) \bmod N],\ 1\big)\,.\end{aligned}$$

By Proposition 15.6(3), this means that $\hat{c} = (1+N)^{[m\cdot\phi(N) \bmod N]} \bmod N^2$. Using Proposition 15.6(2), we know that

$$\hat{c} = (1+N)^{[m\cdot\phi(N) \bmod N]} = (1 + [m \cdot \phi(N) \bmod N] \cdot N) \bmod N^2.$$

Since $1+[m\cdot\phi(N) \bmod N]\cdot N$ is always less that N^2 we can drop the $\bmod N^2$ at the end and view the above as an equality over the integers. Thus, $\hat{m} \stackrel{\text{def}}{=} (\hat{c}-1)/N = [m \cdot \phi(N) \bmod N]$ and, finally,

$$m = [\hat{m} \cdot \phi(N)^{-1} \bmod N],$$

as required. (Note that $\phi(N)$ is invertible modulo N since $\gcd(N, \phi(N)) = 1$.)

CONSTRUCTION 15.11

Let GenModulus be a polynomial-time algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$ and p and q are n-bit primes (except with probability negligible in n). Define the following encryption scheme:

- Gen: on input 1^n run GenModulus(1^n) to obtain (N, p, q). The public key is N, and the private key is $\langle N, \phi(N)\rangle$.
- Enc: on input a public key N and a message $m \in \mathbb{Z}_N$, choose a uniform $r \leftarrow \mathbb{Z}_N^*$ and output the ciphertext

$$c := [(1+N)^m \cdot r^N \bmod N^2].$$

- Dec: on input a private key $\langle N, \phi(N)\rangle$ and a ciphertext c, compute

$$m := \left[\frac{[c^{\phi(N)} \bmod N^2] - 1}{N} \cdot \phi(N)^{-1} \bmod N\right].$$

The Paillier encryption scheme.

Homomorphic Encryption

DEFINITION A public-key encryption scheme (Gen, Enc, Dec) is homomorphic if for all n and all (pk,sk) output by Gen(1^n), it is possible to define groups M; C (depending on pk only) such that:

1. The message space is M, and all ciphertexts output by Enc_{pk} are elements of C. For notational convenience, we write M as an additive group and C as a multiplicative group.

2. For any m1,m2 ϵ M, any c1 output by Enc_{pk}(m1), and any c2 output by Enc_{pk}(m2), it holds that

$$Dec_{sk}(\text{c1.c2}) = \text{m1} + \text{m2}.$$

Moreover, the distribution on ciphertexts obtained by encrypting m1, encrypting m2, and then multiplying the results is identical to the distribution on ciphertexts obtained by encrypting m1 + m2.

2.3 Secret Sharing and Threshold Encryption

Secret Sharing

COROLLARY 15.15 *Any two distinct degree-t polynomials p and q agree on at most t points.*

PROOF If not, then the nonzero, degree-t polynomial $p - q$ would have more than t roots. ■

We now describe Shamir's (t, N)-threshold secret-sharing scheme. Let $\mathbb{F}$ be a finite field that contains the domain of possible secrets, and with $|\mathbb{F}| > N$. Let $x_1, \ldots, x_N \in \mathbb{F}$ be distinct, nonzero elements that are fixed and publicly known. (Such elements exist since $|\mathbb{F}| > N$.) The scheme works as follows:

Sharing: Given a secret $s \in \mathbb{F}$, the dealer chooses uniform $a_1, \ldots, a_{t-1} \in \mathbb{F}$ and defines the polynomial $p(X) \stackrel{\text{def}}{=} s + \sum_{i=1}^{t-1} a_i X^i$. This is a uniform degree-$(t-1)$ polynomial with constant term s. The share of user P_i is $s_i := p(x_i) \in \mathbb{F}$.

Reconstruction: Say t users $P_{i_1}, \ldots, P_{i_t}$ pool their shares $s_{i_1}, \ldots, s_{i_t}$. Using polynomial interpolation, they compute the unique degree-$(t-1)$ polynomial p' for which $p'(x_{i_j}) = s_{i_j}$ for $1 \le j \le t$. The secret is $p'(0)$.

It is clear that reconstruction works since $p' = p$ and $p(0) = s$.

It remains to show that any $t - 1$ users learn nothing about the secret s from their shares. By symmetry, it suffices to consider the shares of users $P_1, \ldots, P_{t-1}$. We claim that for *any* secret s, the shares $s_1, \ldots, s_{t-1}$ are (jointly) uniform. Since the dealer chooses $a_1, \ldots, a_{t-1}$ uniformly, this follows if we show that there is a one-to-one correspondence between the polynomial p chosen by the dealer and the shares $s_1, \ldots, s_{t-1}$. But this is a direct consequence of Corollary 15.15.

Verifiable Secret Sharing

The sharing phase now involves the N users running an interactive protocol with the dealer that proceeds as follows:

1. To share a secret s, the dealer chooses uniform $a_0 \in \mathbb{Z}_q$ and then shares a_0 as in Shamir's scheme. That is, the dealer chooses uniform $a_1, \ldots, a_{t-1} \in \mathbb{Z}_q$ and defines the polynomial $p(X) \stackrel{\text{def}}{=} \sum_{j=0}^{t-1} a_j X^j$. The dealer sends the share $s_i := p(i) = \sum_{j=0}^{t-1} a_j \cdot i^j$ to user P_i.[3]

   In addition, the dealer publicly broadcasts the values $A_0 := g^{a_0}$, $\ldots$, $A_{t-1} := g^{a_{t-1}}$, and the "masked secret" $c := H(a_0) \oplus s$.

2. Each user P_i verifies that its share s_i satisfies

$$g^{s_i} \stackrel{?}{=} \prod_{j=0}^{t-1} (A_j)^{i^j}. \tag{15.3}$$

   If not, P_i publicly broadcasts a complaint.

   Note that if the dealer is honest, we have

$$\prod_{j=0}^{t-1} (A_j)^{i^j} = \prod_{j=0}^{t-1} \left(g^{a_j}\right)^{i^j} = g^{\sum_{j=0}^{t-1} a_j \cdot i^j} = g^{p(i)} = g^{s_i},$$

   and so no honest user will complain. Since there are at most $t-1$ corrupted users, there are at most $t-1$ complaints if the dealer is honest.

3. If more than $t-1$ users complain, the dealer is disqualified and the protocol is aborted. Otherwise, the dealer responds to a complaint from P_i by broadcasting s_i. If this share does not satisfy Equation (15.3) (or if the dealer refuses to respond to a complaint at all), the dealer is disqualified and the protocol is aborted. Otherwise, P_i uses the broadcast value (rather than the value it received in the first round) as its share.

2.4 The Gold wasser-Micali Encryption Scheme

Quadratic Residues Moduloa Prime

In a group G, an element y ϵ G is a quadratic residue if there exists an x ϵ G with x^2 = y. In this case, we call x a square root of y. An element that is not a quadratic residue is called a quadratic non-residue. In an abelian group, the set of quadratic residues forms a subgroup.

PROPOSITION

Let p $\gg$ 2 be prime. Every quadratic residue in Z_p^* has exactly two square roots.

PROOF This follows from Theorem 9.66, but we give a direct proof here. Let $y \in \mathbb{Z}_p^*$ be a quadratic residue. Then there exists an $x \in \mathbb{Z}_p^*$ such that $x^2 = y \bmod p$. Clearly, $(-x)^2 = x^2 = y \bmod p$. Furthermore, $-x \neq x \bmod p$: if $-x = x \bmod p$ then $2x = 0 \bmod p$, which implies $p \,|\, 2x$. Since p is prime, this would mean that either $p \,|\, 2$ (which is impossible since $p > 2$) or $p \,|\, x$ (which is impossible since $0 < x < p$). So, $[x \bmod p]$ and $[-x \bmod p]$ are distinct elements of $\mathbb{Z}_p^*$, and y has at least two square roots.

Let $x' \in \mathbb{Z}_p^*$ be a square root of y. Then $x^2 = y = (x')^2 \bmod p$, implying that $x^2 - (x')^2 = 0 \bmod p$. Factoring the left-hand side we obtain

$$(x - x')(x + x') = 0 \bmod p\,,$$

so that (by Proposition 9.3) either $p \,|\, (x - x')$ or $p \,|\, (x + x')$. In the first case, $x' = x \bmod p$ and in the second case $x' = -x \bmod p$, showing that y indeed has only $[\pm x \bmod p]$ as square roots. ■

Let $\mathsf{sq}_p : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$ be the function $\mathsf{sq}_p(x) \stackrel{\text{def}}{=} [x^2 \bmod p]$. The above shows that sq_p is a two-to-one function when $p > 2$ is prime. This immediately implies that *exactly half the elements of $\mathbb{Z}_p^*$ are quadratic residues.* We denote the set of quadratic residues modulo p by $\mathcal{QR}_p$, and the set of quadratic non-residues by $\mathcal{QNR}_p$. We have just seen that for $p > 2$ prime

$$|\mathcal{QR}_p| = |\mathcal{QNR}_p| = \frac{|\mathbb{Z}_p^*|}{2} = \frac{p-1}{2}\,.$$

PROPOSITION 15.17 *Let $p > 2$ be a prime. Then $\mathcal{J}_p(x) = x^{\frac{p-1}{2}} \bmod p$.*

PROOF Let g be an arbitrary generator of $\mathbb{Z}_p^*$. If x is a quadratic residue modulo p, our earlier discussion shows that $x = g^i$ for some even integer i. Writing $i = 2j$ with j an integer we then have

$$x^{\frac{p-1}{2}} = \left(g^{2j}\right)^{\frac{p-1}{2}} = g^{(p-1)j} = \left(g^{p-1}\right)^j = 1^j = 1 \bmod p\,,$$

and so $x^{\frac{p-1}{2}} = +1 = \mathcal{J}_p(x) \bmod p$ as claimed.

On the other hand, if x is not a quadratic residue then $x = g^i$ for some odd integer i. Writing $i = 2j + 1$ with j an integer, we have

$$x^{\frac{p-1}{2}} = \left(g^{2j+1}\right)^{\frac{p-1}{2}} = \left(g^{2j}\right)^{\frac{p-1}{2}} \cdot g^{\frac{p-1}{2}} = 1 \cdot g^{\frac{p-1}{2}} = g^{\frac{p-1}{2}} \bmod p.$$

Now,

$$\left(g^{\frac{p-1}{2}}\right)^2 = g^{p-1} = 1 \bmod p,$$

and so $g^{\frac{p-1}{2}} = \pm 1 \bmod p$ since $[\pm 1 \bmod p]$ are the two square roots of 1 (cf. Proposition 15.16). Since g is a generator, it has order $p - 1$ and so $g^{\frac{p-1}{2}} \neq 1 \bmod p$. It follows that $x^{\frac{p-1}{2}} = -1 = \mathcal{J}_p(x) \bmod p$. ■

Quadratic Residues Moduloa Composite

Proposition 15.17 directly gives a polynomial-time algorithm (cf. Algorithm 15.18) for testing whether an element $x \in \mathbb{Z}_p^*$ is a quadratic residue.

ALGORITHM 15.18
Deciding quadratic residuosity modulo a prime

Input: A prime p; an element $x \in \mathbb{Z}_p^*$
Output: $\mathcal{J}_p(x)$ (or, equivalently, whether x is a quadratic residue or quadratic non-residue)

$b := \left[x^{\frac{p-1}{2}} \bmod p\right]$
if $b = 1$ **return** "quadratic residue"
else return "quadratic non-residue"

PROPOSITION 15.19 *Let $p > 2$ be a prime, and $x, y \in \mathbb{Z}_p^*$. Then*

$$\mathcal{J}_p(xy) = \mathcal{J}_p(x) \cdot \mathcal{J}_p(y).$$

PROOF Using the previous proposition,

$$\mathcal{J}_p(xy) = (xy)^{\frac{p-1}{2}} = x^{\frac{p-1}{2}} \cdot y^{\frac{p-1}{2}} = \mathcal{J}_p(x) \cdot \mathcal{J}_p(y) \bmod p.$$

Since $\mathcal{J}_p(xy), \mathcal{J}_p(x), \mathcal{J}_p(y) = \pm 1$, equality holds over the integers as well. ■

COROLLARY 15.20 *Let $p > 2$ be prime, and say $x, x' \in \mathcal{QR}_p$ and $y, y' \in \mathcal{QNR}_p$. Then:*

1. $[xx' \bmod p] \in \mathcal{QR}_p$.
2. $[yy' \bmod p] \in \mathcal{QR}_p$.
3. $[xy \bmod p] \in \mathcal{QNR}_p$.

PROPOSITION 15.24 *Let $N = pq$ be a product of distinct, odd primes, and $x, y \in \mathbb{Z}_N^*$. Then $\mathcal{J}_N(xy) = \mathcal{J}_N(x) \cdot \mathcal{J}_N(y)$.*

PROOF Using the definition of $\mathcal{J}_N(\cdot)$ and Proposition 15.19:

$$\begin{aligned}\mathcal{J}_N(xy) = \mathcal{J}_p(xy) \cdot \mathcal{J}_q(xy) &= \mathcal{J}_p(x) \cdot \mathcal{J}_p(y) \cdot \mathcal{J}_q(x) \cdot \mathcal{J}_q(y) \\ &= \mathcal{J}_p(x) \cdot \mathcal{J}_q(x) \cdot \mathcal{J}_p(y) \cdot \mathcal{J}_q(y) = \mathcal{J}_N(x) \cdot \mathcal{J}_N(y).\end{aligned}$$

■

2.5 The Rabin Encryption Scheme

ALGORITHM 15.31
Computing square roots modulo a prime

Input: Prime p; quadratic residue $a \in \mathbb{Z}_p^*$
Output: A square root of a

case $p = 3 \bmod 4$:
  return $[a^{\frac{p+1}{4}} \bmod p]$
case $p = 1 \bmod 4$:
  let b be a quadratic non-residue modulo p
  compute $\ell \geq 1$ and odd m with $2^\ell \cdot m = \frac{p-1}{2}$
  $r := 2^\ell \cdot m,\ r' := 0$
  for $i = \ell$ to 1 {
  // maintain the invariant $a^r \cdot b^{r'} = 1 \bmod p$
    $r := r/2,\ r' := r'/2$
    if $a^r \cdot b^{r'} = -1 \bmod p$
      $r' := r' + 2^\ell \cdot m$
  }
  // now $r = m$, r' is even, and $a^r \cdot b^{r'} = 1 \bmod p$
  return $\left[a^{\frac{r+1}{2}} \cdot b^{\frac{r'}{2}} \bmod p\right]$

That is, to compute a square root of a modulo an integer $N = pq$ of known factorization:

- Compute $a_p := [a \bmod p]$ and $a_q := [a \bmod q]$.
- Using Algorithm 15.31, compute a square root x_p of a_p modulo p and a square root x_q of a_q modulo q.
- Convert from the representation $(x_p, x_q) \in \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ to $x \in \mathbb{Z}_N^*$ with $x \leftrightarrow (x_p, x_q)$. Output x, which is a square root of a modulo N.

It is easy to modify the algorithm so that it returns all four square roots of a.

3 Preliminaries

Definition (responsive, poly-time) We call an executable interactive system IS responsive if Pr [IS(κ) = inf] = 0 for all κ.
Definition We say that (G,E,D) is IND-CCA secure if A_0 comp $\equiv A_1$. We say that (G,E,D) is IND-CPA secure if $A_0^{'} \equiv A_1^{'}$, where $A_b^{'}$ is A_b with the ports dec and plaintext removed.

CONSTRUCTION 15.39

Let GenModulus be a polynomial-time algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$ and p and q are n-bit primes (except with probability negligible in n) with $p = q = 3 \bmod 4$. Construct a public-key encryption scheme as follows:

- Gen: on input 1^n run GenModulus(1^n) to obtain (N, p, q). The public key is N, and the private key is $\langle p, q \rangle$.
- Enc: on input a public-key N and message $m \in \{0, 1\}$, choose a uniform $x \in \mathcal{QR}_N$ subject to the constraint that $\mathsf{lsb}(x) = m$. Output the ciphertext $c := [x^2 \bmod N]$.
- Dec: on input a private key $\langle p, q \rangle$ and a ciphertext c, compute the unique $x \in \mathcal{QR}_N$ such that $x^2 = c \bmod N$, and output $\mathsf{lsb}(x)$.

The Rabin encryption scheme.

PROPOSITION 2.19 *The following holds for all interactive systems $\mathcal{IS}_0, \mathcal{IS}_1, \mathcal{IS}_2$ and all classes of environments* Env.

1. $\mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_0$.
2. *If* $\mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_1$, *then* $\mathcal{IS}_1 \overset{\text{Env}}{\equiv} \mathcal{IS}_0$.
3. *If* $\mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_1$ *and* $\mathcal{IS}_1 \overset{\text{Env}}{\equiv} \mathcal{IS}_2$, *then* $\mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_2$.
4. *If* $\mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_1$ *and* $\mathcal{IS}_2 \in \text{Env}^{\diamond}$, *then* $\mathcal{IS}_2 \diamond \mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_2 \diamond \mathcal{IS}_1$.
5. *If* $\mathcal{IS}_0 \overset{\text{Env}_1}{\equiv} \mathcal{IS}_1$ *and* $\mathcal{IS}_2 \diamond \text{Env}_2 \subseteq \text{Env}_1$, *then* $\mathcal{IS}_2 \diamond \mathcal{IS}_0 \overset{\text{Env}_2}{\equiv} \mathcal{IS}_2 \diamond \mathcal{IS}_1$.

The same properties hold for $\overset{\text{perf}}{\equiv}$, $\overset{\text{stat}}{\equiv}$ *and* $\overset{\text{comp}}{\equiv}$.

4 MPC Protocols with Passive Security

Secret Sharing

Here we concentrate on a particular example scheme that will be sufficient for our purposes in this chapter, namely Shamir's secret sharing scheme. This scheme is based on polynomials over a finite field F. The only necessary restriction on F is that $|F| \gg$ n, but we will assume for concreteness and simplicity that F = Z_p for some prime p $\gg$ n.

A value s ϵ F is shared by choosing a random polynomial f_s(X) ϵ F[X] of degree at most t such that $f_s(0)$ = s. And then sending privately to player P_j the share $s_j = f_s$(j). The basic facts about this method are that any set of t or fewer shares contain no information on s, whereas it can easily be reconstructed from any t+1 or more shares. Both of these facts are proved using Lagrange interpolation.

Lagrange Interpolation

If $h(\mathtt{X})$ is a polynomial over $\mathbb{F}$ of degree at most l and if C is a subset of $\mathbb{F}$ with $|C| = l + 1$, then

$$h(\mathtt{X}) = \sum_{i \in C} h(i)\delta_i(\mathtt{X}) \ ,$$

where $\delta_i(\mathtt{X})$ is the degree l polynomial such that, for all $i, j \in C$, $\delta_i(j) = 0$ if $i \neq j$ and $\delta_i(j) = 1$ if $i = j$. In other words,

$$\delta_i(\mathtt{X}) = \prod_{j \in C, j \neq i} \frac{\mathtt{X} - j}{i - j} \ .$$

Protocol CEPS (Circuit Evaluation with Passive Security)

The protocol proceeds in three phases: the input sharing, computation and output reconstruction phases.

Input Sharing: Each player P_i holding input $x_i \in \mathbb{F}$ distributes $[x_i; f_{x_i}]_t$.

We then go through the circuit and process the gates one by one in the computational order defined above. Just after the input sharing phase, we consider all input gates as being processed. We will maintain the following:

Invariant: Recall that computing with the circuit on inputs $x_1, ..., x_n$ assigns a unique value to every wire. Consider an input or an output wire for any gate, and let $a \in \mathbb{F}$ be the value assigned to this wire. Then, if the gate has been processed, the players hold $[a; f_a]_t$ for a polynomial f_a.

We then continue with the last two phases of the protocol:

Computation Phase: Repeat the following until all gates have been processed (then go to the next phase): Consider the first gate in the computational order that has not been processed yet. According to the type of gate, do one of the following

Addition gate: The players hold $[a; f_a]_t, [b; f_b]_t$ for the two inputs a, b to the gate. The players compute $[a; f_a]_t + [b; f_b]_t = [a + b; f_a + f_b]_t$.

Multiply-by-constant gate: The players hold $[a; f_a]_t$ for the inputs a to the gate. The players compute $\alpha[a; f_a]_t = [\alpha a; \alpha f_a]_t$.

Multiplication gate: The players hold $[a; f_a]_t, [b; f_b]_t$ for the two inputs a, b to the gate.

1. The players compute $[a; f_a]_t * [b; f_b]_t = [ab; f_a f_b]_{2t}$.
2. Define $h \stackrel{\text{def}}{=} f_a f_b$. Then $h(0) = f_a(0) f_b(0) = ab$ and the parties hold $[ab; h]_{2t}$, i.e., P_i holds $h(i)$. Each P_i distributes $[h(i); f_i]_t$.
3. Note that $\deg(h) = 2t \leq n - 1$. Let $\mathbf{r}$ be the recombination vector defined in section 3.2, that is, the vector $\mathbf{r} = (r_1, \ldots, r_n)$ for which it holds that $h(0) = \sum_{i=1}^{n} r_i h(i)$ for any polynomial h of degree $\leq n - 1$. The players compute

$$\begin{aligned} \sum_i r_i [h(i); f_i]_t &= [\sum_i r_i h(i); \sum_i r_i f_i]_t \\ &= [h(0); \sum_i r_i f_i]_t = [ab; \sum_i r_i f_i]_t \ . \end{aligned}$$

Output Reconstruction: At this point all gates, including the output gates have been processed. So do the following for each output gate (labeled i): The players hold $[y; f_y]_t$ where y is the value assigned to the output gate. Each P_j securely sends $f_y(j)$ to P_i, who uses Lagrange interpolation to compute $y = f_y(0)$ from $f_y(1), \ldots, f_y(t+1)$, or any other $t+1$ points.

5 Models

5.1 The UC Model

5.1.1 Clock-Driven Execution

initialization A clocked entity holds as part of its state a bit active ϵ 0, 1, which is initially set to 0. When active = 0 the clocked entity is said to be inactive. When active = 1 the clocked entity is said to be active. It also holds a bit calling ϵ 0, 1, which is initially set to 0. When calling = 1 the clocked entity is said to be calling.

activation bounce during inactivity If an inactive clocked entity receives the activation token on any port with a name not of the form N.infl or N.infli, then it returns the activation token on the matching outport without doing anything else, i.e., it does not read messages, it does not change state and it does not send messages.

clocking If a inactive clocked entity receives the activation token on an open inport with a name of the form N.infl or N.infli, then it sets active $\longleftarrow$ 1 and stores the name CP of the inport on which it was activated. We say that it was called on CP.

return or clock An active clocked entity is only allowed to send the activation token on an outport matching the inport CP on which it was called or an open outport with a name of the form R.infl or R.infli.

return the call If an active clocked entity sends the activation token on the outport matching the inport CP on which it was called, then it sets active $\longleftarrow$ 0. In that case we say that it returned the call.

recursive call If an active clocked entity sends the activation token on an open outport named R.infl or R.infli, it first sets calling $\longleftarrow$ 1 and stores the name RP of the port on which it did the recursive call. We say that it did a recursive call on RP.

activation bounce during calls If a calling clocked entity receives the activation token any inport P which does not match the outport RP on which it did the recursive call, then it returns the activation token on the outport matching P without doing anything else.

result If a calling clocked entity receives the activation token on the inport matching the outport RP on which it did the recursive call, then it sets calling $\longleftarrow$ 0.

Together, these rules ensure that all clocked entities pass the activation token in a simple recursive manner over matching influence and leakage ports, and that they only do work when they are called on their influence ports or after having being returned a recursive call.

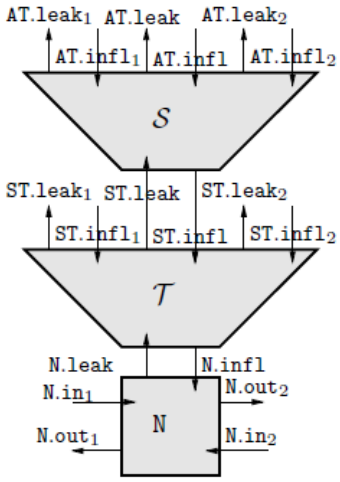
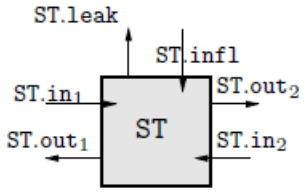


Figure 3: Diagram used for explaining the rules for activation.

5.1.2 Ideal Functionalities

Agent F_{ST}

The ideal functionality for secure transfer between n parties.



- On input (mid, j, m) on $\texttt{ST.in}_i$ (where $j \in \{1, \ldots, n\}$), output $(mid, i, j, |m|)$ on `ST.leak` and store (mid, i, j, m). Ignore any later input of the form $(mid, j, \cdot)$ on $\texttt{ST.in}_i$. Here mid is a message identifier used to distinguish different messages from i to j.
- On input $(\texttt{deliver}, mid, i, j)$ on `ST.infl`, where some (mid, i, j, m) is stored, delete (mid, i, j, m) and output (mid, i, m) on $\texttt{ST.out}_j$.

5.2 Adversaries and Their Powers

5.2.1 Threshold Security

Our protocol Protocol CEPS (Circuit Evaluation with Passive Security) is only secure against t ||| n/2 corrupted parties, as n/2 parties have enough shares in all secret sharings to allow them to reconstruct. This is called threshold security, and in this case the threshold is n/2 1. For any t we

Modeling Corruption

The special ports of an ideal functionality are also used to model which information is allowed to leak when a party is corrupted and which control an adversary gets over a party when that party is corrupted. There are many choices, but we will assume that all ideal functionalities have the following standard corruption behavior.

- On input $(\texttt{passive corrupt}, i)$ on $\mathsf{F}.\texttt{infl}$, the ideal functionality F outputs $(\texttt{state}, i, \sigma)$ on $\mathsf{F}.\texttt{leak}$, where σ is called the ideal internal state of party i and is defined to be all previous inputs on $\mathsf{F}.\texttt{in}_i$ plus those on the message queue of $\mathsf{F}.\texttt{in}_i$, along with all previous outputs on $\mathsf{F}.\texttt{out}_i$.[5]
- On input $(\texttt{active corrupt}, i)$ on $\mathsf{F}.\texttt{infl}$, the ideal functionality F records that the party i is corrupted and outputs the ideal internal state of party i on $\mathsf{F}.\texttt{leak}$. Then it starts ignoring all inputs on $\mathsf{F}.\texttt{in}_i$ and stops giving outputs on $\mathsf{F}.\texttt{out}_i$. Instead, whenever it gets an input $(\texttt{input}, i, x)$ on $\mathsf{F}.\texttt{infl}$, it behaves exactly as if x had arrived on $\mathsf{F}.\texttt{in}_i$, and whenever F is about to output some value y on $\mathsf{F}.\texttt{out}_i$, it instead outputs $(\texttt{output}, i, y)$ on $\mathsf{F}.\texttt{leak}$.

let Envt be the set of Z Env which corrupts at most t parties. To prove that this is an environment class, the crucial observation is that if Z ϵ Envt and S ϵ Sim, then also Z o S corrupts at most t parties, as S is corruption preserving.

5.2.2 Adaptive Security versus Static Security

Environments in Env are allowed to corrupt parties when they desire. This is called adaptive corruption and the environment is called an adaptive adversary, as it can adapt its corruption pattern to the communication that it observes in the protocol. Protocols which can be proven secure against adaptive adversaries are called adaptively secure.

5.2.3 Active Security versus Passive Security

protocols can be secure against passive corruptions but not active corruptions, where the environment takes complete control over the corrupted party. By definition environments Z ϵ Env are allowed active corruptions. We use $Env^{passive}$ to denote the set of Z ϵ Env which only does passive corruptions. That it is an environment class follows from all simulators being corruption preserving. This defines notions of a passive adversary and passively secure and active adversary and active secure. We also call a protocol which is active secure robust and we call a protocol which is passively secure private.

DEFINITION 4.18 *The simulator $\mathcal{S}$ is* conversation-based *if the following is satisfied:*

Conversation-based inputs: *If $\mathcal{S}$ sends an input x to F on behalf of P_i in round j, it computes x as $x = \mathrm{Inp}_i(c)$ where Inp_i is a function specified by the protocol and $c = \mathrm{Conv}_{P_i}(\mathcal{Z} \diamond \mathcal{S} \diamond F)_j$.*

Honest behavior implies correct inputs: *If P_i is corrupt but has followed the protocol honestly then it is always the case that $\mathrm{Inp}_i(c_i)$ equals the corresponding input P_i was given from the environment. By a corrupt P_i following the protocol honestly, we mean that the environment decides the actions of P_i by running a copy of the code of the honest P_i on the inputs and messages that P_i receives in the protocol*[14].

Corruption-consistent input functions: *Consider a conversation c of P_i. Form a new conversation c' by deleting the messages exchanged with some of the honest players, but such that at least $n - t$ honest players remain. For all such c, c' and all input functions, it must be the case that $\mathrm{Inp}_i(c) = \mathrm{Inp}_i(c')$.*

6 Information Theoretic Robust MPC Protocols

6.1 Model for Homomorphic Commitments and some Auxiliary Protocols

Some general remarks on the definition of F_{COM}-

- When we say that a command gives a particular output, then this output is not delivered immediately. Instead it will be specified via COM.infl in which round to deliver, and in that round the output to all parties will be given, i.e., FCOM guarantees simultaneous output with adversarially chosen output round
- If a party Pi is corrupted in between a command is given to FCOM and the output delivery round, then we allow that the input of Pi to the command is changed via COM.infl, and the outputs will be recomputed accordingly. As an example, if Pi becomes corrupted in commit before the output is delivered, then we allow the environment to delete the input of Pi, thereby making the output (commit, i, cid, fail). This models that if a committer Pi becomes (adaptively) corrupted during the implementation of the commitment protocol, it is allowed that the now corrupted Pi can make the protocol fail. This is important for being able to prove adaptive security: Requiring that the protocol will always terminate correctly if just the first round was run honestly is often impossible to implement.
- When we say that a command leaks a value, then the value is output on COM.leak in the round where inputs to the command are given, not in the delivery round. For a discussion of this so-called rushing behavior

6.2 IC Signatures

Protocol TRANSFER

If any command fails in Steps 1-3, it will be clear that either P_i or P_j is corrupt. In this case, go to Step 4

1. $(\mathsf{P}_j)a \leftarrow \langle a \rangle_i$.
2. $\langle a \rangle_j \leftarrow a$.
3. For $k = 1, \ldots, n$ do:[a]
   1. P_i picks a uniformly random $r \in_R \mathbb{F}$ and sends it privately to P_j.
   2. Execute $\langle r \rangle_i \leftarrow r$ and $\langle r \rangle_j \leftarrow r$.
   3. P_k broadcasts a random challenge $e \in \mathbb{F}$.
   4. The parties execute $\langle s \rangle_i \leftarrow e\langle a \rangle_i + \langle r \rangle_i$ and $\langle s \rangle_j \leftarrow e\langle a \rangle_j + \langle r \rangle_j$.
   5. $s \leftarrow \langle s \rangle_i$ and $s' \leftarrow \langle s \rangle_j$ (we use s' here to indicate that the opened values may be different if P_i or P_j is corrupt).
   6. If $s \neq s'$, all players go to step 4.

   If all n iterations of the loop were successful, parties output `success`, except P_j who outputs a.
4. This is the "exception handler". It is executed if it is known that P_j or P_i is corrupt. First execute $a \leftarrow \langle a \rangle_i$. If this fails, all parties output `fail`. Otherwise do $\langle a \rangle_j \Leftarrow a$ (this cannot fail). Parties output `success`, except P_j who outputs a.

[a] The description of this step assumes that $1/|\mathbb{F}|$ is negligible in the security parameter. If that is not the case, we repeat the step u times in parallel, where u is chosen such that $(1/|\mathbb{F}|)^u$ is negligible.

Protocol PERFECT TRANSFER

If any command fails in Steps 1-4, it will be clear that either P_i or P_j is corrupt. In this case, go to Step 5

1. $(\mathsf{P}_j)a \leftarrow \langle a \rangle_i$.
2. $\langle a \rangle_j \leftarrow a$.
3. To check that P_i and P_j are committed to the same value, we do the following:
   P_i selects a polynomial $f(\mathtt{X}) = a + \alpha_1 \mathtt{X} + \cdots + \alpha_t \mathtt{X}^t$, for uniformly random $\alpha_1, \ldots, \alpha_t$, and executes $\langle \alpha_l \rangle_i \leftarrow \alpha_l$ for $l = 1, \ldots, t$. We invoke the `add` and `mult` commands so all players collaborate to compute, for $k = 1, \ldots, n$:

   $$\langle f(k) \rangle_i = \langle a \rangle_i + k\langle \alpha_1 \rangle_i + \cdots + k^t \langle \alpha_t \rangle_i$$

   and then $(\mathsf{P}_k) f(k) \leftarrow \langle f(k) \rangle_i$.
   Then P_i sends $\alpha_1, \ldots, \alpha_t$ privately to P_j and we execute $\langle \alpha_l \rangle_j \leftarrow \alpha_l$ for $l = 1, \ldots, t$. In the same way as above, we compute $\langle f(k) \rangle_j$ and execute $(\mathsf{P}_k) f(k) \leftarrow \langle f(k) \rangle_j$.
4. For $k = 1, \ldots, n$: P_k compares the two values that were opened towards him. If they agree, he broadcasts `accept`, else he broadcasts `reject`.
   For every P_k who said reject, execute $a_k \leftarrow \langle f(k) \rangle_i$ and $a'_k \leftarrow \langle f(k) \rangle_j$. If $a_k = a'_k$ for all k, all players output `success`, except P_j who outputs a. Otherwise go to step 5.
5. This is the "exception handler". It is executed if it is known that P_j or P_i is corrupt. First execute $a \leftarrow \langle a \rangle_i$. If this fails, all parties output `fail`. Otherwise do $\langle a \rangle_j \Leftarrow a$ (this cannot fail). Parties output `success`, except P_j who outputs a.

Protocol COMMITMENT MULTIPLICATION

If any command used during this protocol fails, all players output `fail`.

1. $\mathsf{P}_i : \langle c \rangle_i \leftarrow ab$.
2. For $k = 1, \ldots, n$, do:[a]
   1. P_i chooses a uniformly random $\alpha \in \mathbb{F}$.
   2. $\langle \alpha \rangle_i \leftarrow \alpha$; $\langle \gamma \rangle_i \leftarrow \alpha b$.
   3. P_k broadcasts a uniformly random challenge $e \in \mathbb{F}$.
   4. $\langle A \rangle_i \leftarrow e\langle a \rangle_i + \langle \alpha \rangle_i$; $A \leftarrow \langle A \rangle_i$.
   5. $\langle D \rangle_i \leftarrow A\langle b \rangle_i - e\langle c \rangle_i - \langle \gamma \rangle_i$; $D \leftarrow \langle D \rangle_i$.
   6. If $D \neq 0$ all players output `fail`.
3. If we arrive here, no command failed during the protocol and all D-values were 0. All players output `success`.

[a] The description of this step assumes that $1/|\mathbb{F}|$ is negligible in the security parameter. If that is not the case, we repeat the step u times in parallel, where u is chosen such that $(1/|\mathbb{F}|)^u$ is negligible.

Protocol PERFECT COMMITMENT MULTIPLICATION

If any command used during this protocol fails, all players output `fail`.

1. $\langle c \rangle_i \leftarrow ab$.
2. P_i chooses polynomials $f_a(\mathtt{X}) = a + \alpha_1 \mathtt{X} + \cdots + \alpha_t \mathtt{X}^t$ and $f_b(\mathtt{X}) = b + \beta_1 \mathtt{X} + \cdots + \beta_t \mathtt{X}^t$, for random α_j, β_j. He computes $h(\mathtt{X}) = f_a(\mathtt{X}) f_b(\mathtt{X})$, and writes $h(\mathtt{X})$ as $h(\mathtt{X}) = c + \gamma_1 \mathtt{X} + \cdots + \gamma_{2t} \mathtt{X}^{2t}$. Then establish commitments as follows:

$$\langle \alpha_j \rangle_i \leftarrow \alpha_j \text{ for } j = 1, \ldots, t$$
$$\langle \beta_j \rangle_i \leftarrow \beta_j \text{ for } j = 1, \ldots, t$$
$$\langle \gamma_j \rangle_i \leftarrow \gamma_j \text{ for } j = 1, \ldots, 2t$$

3. The `add` and `mult` commands are invoked to compute, for $k = 1, \ldots, n$:

$$\langle f_a(k) \rangle_i = \langle a \rangle_i + k\langle \alpha_1 \rangle_i + \cdots + k^t \langle \alpha_t \rangle_i$$
$$\langle f_b(k) \rangle_i = \langle b \rangle_i + k\langle \beta_1 \rangle_i + \cdots + k^t \langle \beta_t \rangle_i$$
$$\langle h(k) \rangle_i = \langle b \rangle_i + k\langle \gamma_1 \rangle_i + \cdots + k^{2t} \langle \gamma_{2t} \rangle_i$$

   Then $(\mathsf{P}_k)a_k \leftarrow \langle f_a(k) \rangle_i$, $(\mathsf{P}_k)b_k \leftarrow \langle f_b(k) \rangle_i$ and $(\mathsf{P}_k)c_k \leftarrow \langle h(k) \rangle_i$ are executed.
4. For $k = 1, \ldots, n$, do: P_k checks that $a_k b_k = c_k$, and broadcasts `accept` if this is the case, else broadcast `reject`.
5. For each P_k who said `reject`, do $a_k \leftarrow \langle f_a(k) \rangle_i$, $b_k \leftarrow \langle f_b(k) \rangle_i$ and $c_k \leftarrow \langle h(k) \rangle_i$, all players check whether $a_k b_k = c_k$ holds. If this is not the case, all players output `fail`.
6. If we arrive here, no command failed during the protocol and all relations $a_k b_k = c_k$ that were checked, hold. All players output `success`.

6.3 Optimality of Corruption Bounds

Theorem- There exists a protocol πBROADCAST such that πBROADCAST - FPPC implements FSC in Env(t,sync) with perfect security for all t $\leq$ n/3.

Protocol ICSig (Continued)

(designated) reveal: On input $(\texttt{reveal}, sid)$ to all players, P_{INT} broadcasts $a, \{m_{i,a}\}_{i=1,\ldots,n}$ as stored under sid. Each P_i retrieves $K_{i.a}$ stored under sid and checks that $m_{i,a} = \mathrm{MAC}_{K_{i,a}}(a)$. If this holds, he outputs $(\texttt{reveal}, sid, a)$ else he output $(\texttt{reveal}, sid, \texttt{fail})$.
If the command was designated, $(\texttt{reveal}, j, sid)$, P_{INT} only sends $a, m_{j,a}$ to P_j who checks the MAC as in the normal `reveal` command.

add: On input $(\texttt{add}, sid_1, sid_2, sid_3)$ to all players, P_{INT} retrieves the values $(a_1, m_{1,a_1}, \ldots, m_{n,a_1}), (a_2, m_{1,a_2}, \ldots, m_{n,a_2})$, that were stored under sid_1, sid_2 and stores $(a_1 + a_2, m_{1,a_1} + m_{1,a_2}, \ldots, m_{n,a_1} + m_{n,a_2})$ under sid_3.
Each P_i retrieves the values $\beta_{i,a_1}, \beta_{i,a_2}$ stored under sid_1, sid_2 and stores $\beta_{i,a_1} + \beta_{i,a_2}$ under sid_3.
All players output $(\texttt{add}, sid_1, sid_3, sid_3, \texttt{success})$.

multiplication by constant: On input $(\texttt{mult}, c, sid_2, sid_3)$ to all players, P_{INT} retrieves the values $(a_2, m_{1,a_2}, \ldots, m_{n,a_2})$, that were stored under sid_2 and stores $(c \cdot a_2, c \cdot m_{1,a_2}, \ldots, c \cdot m_{n,a_2})$ under sid_3.
Each P_i retrieves the value β_{i,a_2} stored under sid_2 and stores $c \cdot \beta_{i,a_2}$ under sid_3.
All players output $(\texttt{mult}, c, sid_2, sid_3, \texttt{success})$.

Theorem- There exists no protocol πBROADCAST such that πBROADCAST - FPPC implements FSC in Env(t,sync) with perfect or statistical security if t $\geq$ n/3.

7 MPC from General Linear Secret Sharing Schemes

General Adversary Structures
Definition An adversary structure A is said to be Q2 if for all A1,A2 ϵ A it holds that A1 U A2 $\neq$ P. It is said to be Q3 if for all A1,A2,A3 ϵ A it holds that A1 U A2 U A3 $\neq$ P.
Multiplicative Secret-Sharing
Lemma Let c_k be the k-th entry in [a; ra]S - [b; rb]S. Then there exist (fixed) indices u, v such that $c_k = a_u b_v$ where a_u is the u-th entry in [a; ra]S, b_v is the v-th entry in [b; rb]S and ϕ(k) = ϕ(u) = ϕ(v).

8 Cryptographic MPC Protocols

The Case of Honest Majority
Theorem If non-committing encryption schemes exist, then there exists a protocol π^f(COMPSEC) such that π^f(COMPSEC) - F_{AT} implements F^f(SFE) in Env(t,sync, poly) with computational security for all t $\leq$ n/2. **The Case of Dishonest Majority**
Theorem Assuming STPs exist, there exists a protocol π^f(COMPSEC-DM) in the CRS model, such that π^f(COMPSEC-DM) - FAT implements $F(f, abort)_{SFE}$ in Env(t,sync,poly(with computational security for all t $\leq$ n.

9 Some Techniques for Efficiency Improvements

9.1 Circuit Randomization

Definition A linear representation [·] over a finite field F satisfies the following properties:

- Any player Pi can collaborate with the other players to create [r], where r F is chosen by Pi. If Pi is honest, the process reveals no information on r. [r] will be correctly formed no matter whether Pi is honest or not.
- If players hold representations [a], [b] and a public constant α, they can locally compute new representations [a] + [b] = [a + b] and α[a] = [αa] of the same form as [a] and [b].
- Given [a], [b], the players can interact to compute a new representation [a][b] = [ab] of the same form as [a] and [b] while revealing nothing new about a, b.
- The players can collaborate to open any representation, in particular also representations of form [a + b] or [ab]. This will reveal a + b, respectively ab but will reveal no other information on a, b.

9.2 Hyper-invertible Matrices

A hyper-invertible matrix M over a finite field F is a matrix with the property that any square submatrix is invertible.
Definition A matrix M is hyper-invertible if the following holds: Let R be a subset of the rows, and let M_R denote the sub-matrix of M consisting of rows in R. Likewise, let C be a subset of columns and let M^C denote the sub-matrix consisting of columns in C. Then we require that M_R^C is invertible whenever $|R| = |C| \geq 0$.

9.3 Packed Secret-Sharing

Lemma Let the secret vector s and random polynomial f_s be defined. Then, any subset of at most t shares has a distribution independent of s and from any set of at least l + t shares, one can reconstruct s. **Theorem** πON-LINE - F(TRIP) implements F_{SFE}(f,abort) in Env(t,sync,static) with statistical security for all t $\leq$ n.

10 Applications of MPC

A Double Auction
Various types of auctions- This is not limited to only standard highest bid auctions with sealed bids but also includes, for instance, variants with many sellers and buyers, so-called double auctions: essentially scenarios where one wants to find a fair market price for a commodity given the existing supply and demand in the market.
Benchmarking- where several companies want to combine information on how their businesses are running, in order to compare themselves to best practice in the area. The benchmarking process is either used for learning, planning or motivation purposes. This of course has to be done while preserving confidentiality of companies' private data.

Protocol COMPARE

Input: Sharings $[a]$ and $[b]$.
Output: A sharing $[c]$ where $c = 1$ if $a > b$ and $c = 0$ otherwise.
Algorithm: 1. $([a_\ell], \ldots, [a_0]) = \text{BITS}([a])$.
2. $([b_\ell], \ldots, [b_0]) = \text{BITS}([b])$.
3. For $i = 0, \ldots, \ell$: $[c_i] = [a_i] + [b_i] - 2[a_i][b_i]$.
4. $([d_\ell], \ldots, [d_0]) = \text{MS1}([c_\ell], \ldots, [c_0])$.
5. For $i = 0, \ldots, \ell$: $[e_i] = [a_i][d_i]$.
6. $[c] = \sum_{i=0}^{\ell} [e_i]$.

Protocol MS1

Input: Sharings $[c_\ell], \ldots, [c_0]$ of bits.
Output: Sharings $[d_\ell], \ldots, [d_0]$, where $d_\ell \cdots d_0 = \text{MS1}(c_\ell \cdots c_0)$.
Algorithm: 1. Let $[f_{\ell+1}]$ be a dummy sharing of 1.
2. For $i = \ell, \ldots, 0$: $[f_i] \leftarrow [f_{i+1}](1 - [c_i])$.
3. For $i = \ell, \ldots, 0$: $[d_i] \leftarrow [f_{i+1}] - [f_i]$.

Protocol BITS

Input: A sharing $[a]$ of $a \in \mathbb{Z}_p$.
Output: Sharings $[f_\ell], \ldots, [f_0]$ of the bits of a.
Algorithm: 1. $([r], [r_\ell], \ldots, [r_0]) \leftarrow$ RANDOMSOLVEDBITS.
2. $[c] = [a] - [r]$.
3. $c \leftarrow$ RECONSTRUCT$([c])$.
4. Write c in binary $c_\ell, \ldots, c_0$.
5. $([d_{\ell+1}], \ldots, [d_0]) \leftarrow$ BITADD$(([r_\ell], \ldots, [r_0]), (c_\ell, \ldots, c_0))$.
6. Write p in binary $(p_{\ell+1}, p_\ell, \ldots, p_0)$.
7. $[e] =$ BITCOMPARE$(([d_{\ell+1}], \ldots, [d_0]), (p_{\ell+1}, \ldots, p_0))$.
8. For $i = 0, \ldots, \ell + 1$: $[ep_i] = [e][p_i]$.
9. $([f_{\ell+1}], \ldots, [f_0]) \leftarrow$ BITSUB$(([d_{\ell+1}], \ldots, [d_0]), ([ep_{\ell+1}], \ldots, [ep_0]))$.

Protocol RANDOMSOLVEDBITS

Output: Sharing $[r]$ of a random unknown field element $r \in \mathbb{Z}_p$.
Output: Sharings $[r_\ell] \ldots, [r_0]$ of the bits of r.
Algorithm: 1. For $i = 0, \ldots, \ell$: $[r_i] \leftarrow$ RANDOMBIT().
2. Write p in binary $p_\ell, \ldots, p_0$.
3. $[c] \leftarrow$ BITCOMPARE$((p_\ell, \ldots, p_0), ([r_\ell], \ldots, [r_0]))$.
4. $c \leftarrow$ RECONSTRUCT$([c])$.
5. If $c = 0$, go to Step 1
6. $[r] \leftarrow \sum_{i=0}^{\ell} 2^i [r_i]$.

11 Algebraic Preliminaries

In this module, I have learned about the Groups, Rings, fields, Module and Vector Spaces. Also learned about the direct sum and products and also learned about the tensor products.

Protocol RandomBit

Output: A sharing $[r]$ of a uniformly random bit r.
Algorithm: 1. $[a] \leftarrow$ RandomFieldElement.
2. $[a^2] = [a][a]$.
3. $A \leftarrow$ Reconstruct$([a^2])$
4. If $A = 0$, go to Step 1.
5. $b \leftarrow \sqrt{A} \bmod p$.
6. $[c] = (b^{-1} \bmod p)[a]$.
7. $[r] = 2^{-1}([c] + 1)$.

12 Secret Sharing

In this module, I have learned about Interpolation codes and secret sharing from that and proof of lagranges theorem and also learned about different secret sharing schemes and finally learned the Interpolation over Rings and Blackbox Secret Sharing.