CS 395T: Topics in Cryptography

Topic: Pairings and Lattices

Instructor: David Wu

Compilation of scribe notes from Spring 2024

Warning: These notes are a direct compilation of scribe notes from the Spring 2024 offering of the course. They are only lightly edited for correctness and completeness. There may be typos and errors.

Last updated: April 30, 2024

Contents

Lecture 1: Elliptic Curves 1

Lecture 2: Pairing Groups 5

Lecture 3: Short Signatures 9

Lecture 4: Aggregate and Threshold Signatures 13

Lecture 5: Threshold Signatures 16

Lecture 6: Identity-Based Encryption 20

Lecture 7: Broadcast Encryption 24

Lecture 8: Distributed Broadcast Encryption 28

Lecture 9: Attribute-Based Encryption 34

Lecture 10: Attribute-Based Encryption 40

Lecture 11: Somewhat Homomorphic Encryption 44

Lecture 12: Non-Interactive Zero Knowledge 48

Lecture 13: Batch Arguments 53

Lecture 14: RAM Delegation 57

Lecture 15: Polynomial Commitments 60

Lecture 16: Succinct Non-Interactive Arguments (SNARGs) 64

Lecture 17: Introduction to Lattices 68

Lecture 18: Short Integer Solutions 72

Lecture 19: Lattice Trapdoors 76

Lecture 20: Lattice-based Signatures and Learning with Errors (LWE) 79

Lecture 21: Regev Encryption and Fully Homomorphic Encryption 82

Lecture 22: FHE Bootstrapping and Key Agreement from LWE 86

Lecture 23: Homomorphic Signatures 88

Lecture 24: Homomorphic Commitments 91

Lecture 25: Functional Commitments 95

Lecture 26: Attribute Based Encryption 99

CS 395T: Topics in Cryptography January 16, 2024

Lecture 1: Elliptic Curves

Lecturer: David Wu *Scribe: David Wu*

Let $\mathbb{G}$ be a group of prime order p with generator g. Recall that the discrete log problem over $\mathbb{G}$ says that given (g, h) where $h = g^x$ and $x \xleftarrow{R} \mathbb{Z}_p$, it is computationally hard to compute x. Recall from the first course in cryptography that the hardness of the discrete logarithm problem (and its variants such as the computational Diffie-Hellman and decisional Diffie-Hellman problems) are very useful for building many *public-key* cryptographic primitives (such as public-key encryption, digital signatures, and more). The question then is to identify groups where the discrete log problem is believed to be hard. The primary instantiations considered today are (1) the integers modulo a prime p; and (2) the group of points on an elliptic curve.

Integers modulo a prime p. The first example of a group $\mathbb{G}$ where the discrete log problem is believed to be hard is the group $\mathbb{G} = \mathbb{Z}_p^*$ where p is a (large) prime. Note that the group $\mathbb{Z}_p^*$ has order $p - 1$, which is *not* prime. It is often convenient to work over a group of *prime* order, so we typically take $\mathbb{G}$ to be a subgroup of $\mathbb{Z}_p^*$. For instance, suppose $p = kq + 1$ for some positive integer $k \in \mathbb{N}$ and q is prime (this setting is often called a "Schnorr group.") Then, we can take $\mathbb{G} \subset \mathbb{Z}_p^*$ to be the subgroup of $\mathbb{Z}_p^*$ of order q (i.e., if g is a generator of $\mathbb{Z}_p^*$, then g^k is a generator of $\mathbb{G}$).

When setting parameters for a Schnorr group, the prime p is typically around 2048 bits (when targeting 128 bits of security). This is necessary to resist attacks based on index calculus and the general number field sieve (GNFS) whose running time is roughly $2^{\tilde{O}(\sqrt[3]{\log p})}$, where the $\tilde{O}(\cdot)$ term suppresses constants and $\mathsf{poly}(\log \log p)$ terms. The smaller prime q is set to be 256 bits to resist generic discrete log algorithms (e.g., Pollard rho). The need for such a large prime p is to resist GNFS, and this is a major source of inefficiency when it comes to deploying cryptographic systems that rely on hardness of discrete log over $\mathbb{Z}_p^*$.

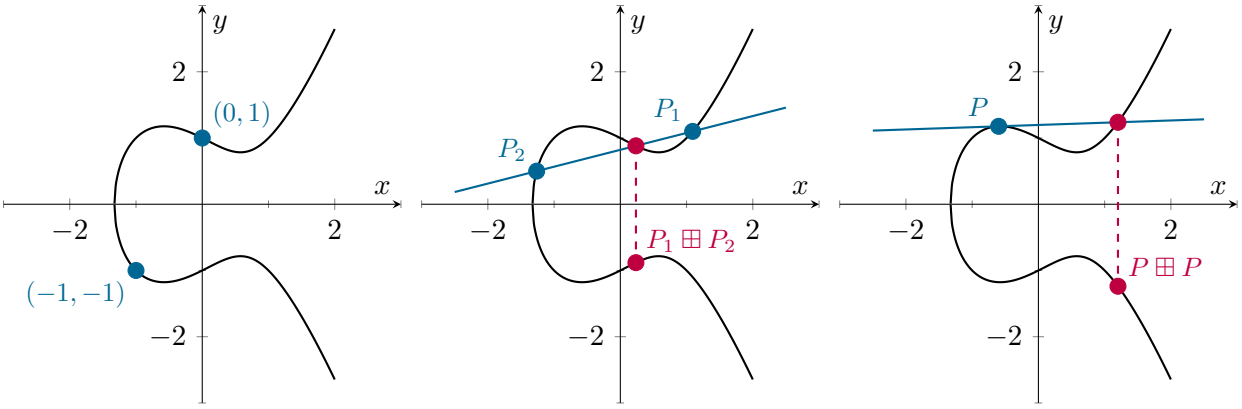
Elliptic curves. An alternative family of groups where the discrete log problem is believed to be hard is to consider the set of points on an elliptic curve. Remarkable, such objects have been studied since antiquity. One of the earliest known examples is from the work of the Greek mathematician Diophantus from the 3rd century AD. Diophantus was specifically interested in the *rational* roots of various bivariate polynomials $f(x, y) = 0$. The particular setting we will be interested in corresponds to the case where $f(x, y)$ is a *cubic* polynomial in the following form:

$$f(x, y) = y^2 - (x^3 + Ax + B),$$

where $A, B \in \mathbb{Q}$ are constants. Here, we view $f \colon \mathbb{R}^2 \to \mathbb{R}$ as a polynomial over the real numbers with rational coefficients. A "rational" solution to $f(x, y)$ is a pair $(x, y) \in \mathbb{Q}^2$ where $f(x, y) = 0$, or equivalently, a pair $(x, y) \in \mathbb{Q}^2$ along the curve $E \colon y^2 = x^3 + Ax + B$. As a concrete example, we plot the curve $y^2 = x^3 - x + 1$ and provide some examples of rational points on the curve in Fig. 1.1a.

The chord method for constructing rational points. Suppose we have two rational points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ on the curve E. Consider the line $y = \alpha x + \beta$, where $\alpha, \beta \in \mathbb{Q}$ that interpolates these two points. It is easy to see that this line will intersect the curve at a third point. Namely, the x-coordinates of the points of intersection are the solutions to the cubic equation

$$f(x) \coloneqq (\alpha x + \beta)^2 - (x^3 + Ax + B).$$



(a) Examples of rational points on E. (b) The chord method. (c) The tangent method.

Figure 1.1: The elliptic curve $E\colon y^2 = x^3 - x + 1$.

This is a cubic equation and two of its roots are $x_1, x_2 \in \mathbb{Q}$. Thus, we can write $f(x) = (x-x_1)(x-x_2)g(x)$, where $g(x)$ is a linear polynomial. Since $f(x)$ has rational coefficients and $x_1, x_2 \in \mathbb{Q}$, the polynomial g also has rational coefficients. Finally, since g is a linear polynomial, it has exactly one root $x_3 \in \mathbb{Q}$. Setting $y_3 = \alpha x_3 + \beta$, we conclude that $(x_3, y_3) \in \mathbb{Q}^2$ is also a rational point on the curve E. This approach provides a general procedure for constructing rational points on an elliptic curve E:

1. Start with two rational points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ where $(x_1, y_1) \neq (x_2, -y_2)$.
2. Construct the line L that interpolates P_1 and P_2.
3. Compute the intersection of L and E. The intersection is a rational point $P_3 = (x_3, y_3)$. Note that it could be the case that $P_3 \in \{P_1, P_2\}$ (i.e., this method is not guaranteed to yield a *new* rational point).

This approach for constructing rational points on an elliptic curve is called the *chord method*.

The tangent method for constructing rational points. A second method for constructing rational points on the curve E is to take a rational point $P = (x, y)$ with $y \neq 0$ and consider the line that is *tangent* to the curve at P. To ensure that the tangent line is well-defined[1] at every point P on the curve, we require that $4A^3 + 27B^2 \neq 0$. The quantity $4A^3 + 27B^2$ is referred to as the *discriminant* of the elliptic curve.

From calculus we see that the slope of the tangent line at P is $(3x^2 + A)/(2y)$, so if $x, y \in \mathbb{Q}$, then the tangent line also has rational coefficients. Let the line $y = \alpha x + \beta$ be the line that is tangent to E at a point $P = (x_1, y_1)$. We argue that the tangent line will intersect the curve at exactly one other (rational) point. By definition, the x-coordinates of the points of intersection are the roots of the polynomial

$$f(x) \coloneqq (\alpha x + \beta)^2 - (x^3 + Ax + B).$$

Since $\alpha x + \beta$ is the tangent line at $P = (x_1, y_1)$, this means $f(x_1) = 0$ and $f'(x_1) = 0$. Since $f(x_1) = 0$, we can write

$$f(x) = (x - x_1) \cdot g(x)$$

[1]Note that we do allow the slope of the tangent line to be ∞, which corresponds to a vertical tangent line.

for some quadratic polynomial $g(x)$. Next, from calculus, we know that

$$f'(x) = g(x) + (x - x_1)g'(x).$$

This means $0 = f'(x_1) = g(x_1)$, which means x_1 is also a root of g. We conclude that

$$f(x) = (x - x_1)g(x) = (x - x_1)^2 h(x),$$

where $h(x)$ is a linear polynomial. Since f has rational coefficients and $x_1 \in \mathbb{Q}$, it follows that the coefficients of h are also rational. Taking $x_2 \in \mathbb{Q}$ to be the root of h and setting $y_2 = \alpha x_2 + \beta$, we obtain a new rational point (x_2, y_2) on the curve E. This approach for constructing rational points on an elliptic curve is called the *tangent method*.

The group law. The chord and tangent methods provides a procedure that takes one or two rational points on an elliptic curve E and outputs another rational point on the curve E. Remarkably, it can be shown that this procedure can be used as the basis of a group law on the set of rational points on the elliptic curve. Namely, for an elliptic curve $E\colon y^2 = x^3 + Ax + B$ with non-zero discriminant, we define $E(\mathbb{Q})$ to be the set of rational points on E together with a special point at infinity $\mathcal{O}$:

$$E(\mathbb{Q}) \coloneqq \left\{(x, y) \in \mathbb{Q}^2 : y^2 = x^3 + Ax + B\right\} \cup \{\mathcal{O}\}.$$

We now proceed to define the group structure over $E(\mathbb{Q})$:

- **Negation:** If $P = (x, y) \in E(\mathbb{Q})$, then we define $-P \coloneqq (x, -y) \in E(\mathbb{Q})$.
- **Identity element:** For all $P \in E(\mathbb{Q})$, we define $P + (-P) = \mathcal{O}$ and $P + \mathcal{O} = P = \mathcal{O} + P$.
- **Group operation:** Given $P_1 = (x_1, y_1) \in E(\mathbb{Q})$ and $P_2 = (x_2, y_2) \in E(\mathbb{Q})$, we define the group operation $P_1 \boxplus P_2$ as follows:
  - If $x_1 \neq x_2$, the "sum" of P_1 and P_2 is obtained by applying the chord method to P_1 and P_2. Let $\tilde{P}_3$ be the rational point output by the chord method. Then, the group operation is defined as $P_1 \boxplus P_2 \coloneqq -\tilde{P}_3$. Visually, this corresponds to drawing the line between P_1 and P_2, finding the third point of intersection $\tilde{P}_3$ with the curve E, and then reflecting $\tilde{P}_3$ about the x-axis. This process is illustrated in Fig. 1.1b. We can also express this algebraically as follows:

    $$x_3 \coloneqq s^2 - x_1 - x_2 \quad \text{and} \quad y_3 \coloneqq s(x_1 - x_3) - y_1 \tag{1.1}$$

    where $s = (y_2 - y_1)/(x_2 - x_1)$ is the slope of the chord.
  - If $x_1 = x_2$ and $y_1 = y_2 \neq 0$ (i.e., $P_1 = P_2$), then the "sum" of P_1 and P_2 is obtained by applying the tangent method to P_1. Let $\tilde{P}_3$ be the rational point output by the tangent method. The group operation is defined as $P_1 \boxplus P_1 \coloneqq -\tilde{P}_3$. Visually, this corresponds to drawing the tangent line to the curve E at point P_1, finding the point $\tilde{P}_3$ where the tangent line intersects E and then reflecting $\tilde{P}_3$ about the x-axis. This process is illustrated in Fig. 1.1c. We can also express this algebraically as follows:

    $$x_3 \coloneqq s^2 - 2x_1 \quad \text{and} \quad y_3 \coloneqq s(x_1 - x_3) - y_1, \tag{1.2}$$

    where $s = (3x_1^2 + A)/(2y_1)$ is the slope of the tangent line.
  - When $x_1 = x_2$ and $y_1 = -y_2$ (i.e., if $P_1 = -P_2$), the output is the identity element $\mathcal{O}$. In this case, the line that passes through P_1 and P_2 is a vertical line; we say that this line intersects the curve "at infinity."[2]

[2] The geometric intuition can be formalized by working in the *projective* plane.

Elliptic curves over finite fields. Thus far, we have described elliptic curves over the rationals. In cryptography, we will work over a finite domain, and instead, consider elliptic curves over a finite field $\mathbb{F}$. First, we introduce the notion of a field:

- A field consists of a set of elements $\mathbb{F}$ equipped with two binary operations: addition (denoted $+$) and multiplication (denoted $\times$).
- The set $\mathbb{F}$ together with the addition operation $+$ is a commutative group. We typically write 0 to denote the additive identity in $\mathbb{F}$.
- The set $\mathbb{F}^\times := \mathbb{F} \setminus \{0\}$ together with the multiplication operation $\times$ is also a commutative group. We typically write 1 to denote the multiplicative identity in $\mathbb{F}$.
- Finally, multiplication distributes over addition: namely, for all $\alpha, x, y \in \mathbb{F}$, it holds that $\alpha(x+y) = \alpha x + \alpha y$.

Notably, every element in a field has an additive inverse, and every non-zero element in a field has a multiplicative inverse. Examples of fields include the real numbers, the rationals, and the integers modulo p where p is a prime. In this course, we consider finite fields. In this case, for a prime p, we will write $\mathbb{F}_p$ to denote the integers modulo p (in this course, we may abuse notation and interchange $\mathbb{Z}_p$ and $\mathbb{F}_p$). We now define the notion of an elliptic curve over $\mathbb{F}_p$. For simplicity, we assume $p > 3$ throughout this course. Let $E\colon y^2 = x^3 + Ax + B$[3] where $A, B \in \mathbb{F}_p$ and $4A^3 + 27B^2 \neq 0$. We write $E(\mathbb{F}_p)$ to denote the group of points on the elliptic curve together with the point at infinity:

$$E(\mathbb{F}_p) := \left\{(x, y) \in \mathbb{F}_p : y^2 = x^3 + Ax + B\right\} \cup \{\mathcal{O}\}\,.$$

The group law is defined using the same *algebraic* relations (specifically, by Eqs. (1.1) and (1.2)) as for the case over the rationals. When working over finite fields, the group law no longer has a clean visual interpretation; nonetheless, the same algebraic expressions are equally well-defined (since $\mathbb{F}_p$ is a field).

Elliptic curves in practice. For many elliptic curves E over a finite field $\mathbb{F}_p$, the best-known algorithm for the discrete logarithm problem is the "generic" one (i.e., a discrete log algorithm that is agnostic to the particular implementation details of the group). In particular, for many curves E, if $|E(\mathbb{F}_p)| = q$ for some prime q, then the best algorithm for discrete log runs in time $O(\sqrt{q})$. Thus, to achieve 128-bits of security, it suffices to consider an elliptic curve group with $\approx 2^{256}$ points (it suffices to work over a finite field of order $p \approx 2^{256}$ to realize this). Importantly, the order of $E(\mathbb{F}_p)$ should not be p; such curves have *embedding degree* 1 and are referred to as "anomalous" curves. As we will see later, for anomalous curves the discrete log problem over $E(\mathbb{F}_p)$ can be reduced to the discrete problem over $\mathbb{F}_p$. In this case, there is no advantage to using elliptic curves as opposed to the integers modulo p.

Today, there are several widely-used elliptic curves that have been standardized. Two prominent examples are `secp256r1` (P-256) and `Curve25519`. Both of these are 256-bit elliptic curves (i.e., the group order is roughly 2^{256}) and are believed to be secure at the 128-bit security level. P-256 was standardized by the National Institute of Standards (NIST) in 1999 while `Curve25519` was designed by Dan Bernstein [Ber06]. Both families of curves are widely used in Diffie-Hellman key agreement. More generally, due to their efficiency, elliptic-curve groups are the most common instantiations for public-key cryptographic systems that rely on the hardness of discrete log.

[3] This is often referred to as the Weierstrass representation of an elliptic curve. For some elliptic curves, there are alternative (and equivalent) ways to describe the curve equation and the group law that can be more amenable for computation. These include the Montgomery representation and the Edwards representation. We will not cover these alternative representations in this course.

CS 395T: Topics in Cryptography January 18, 2024

Lecture 2: Pairing Groups

Lecturer: David Wu *Scribe: David Wu*

Beyond their direct application as a group where the discrete log problem is difficult, the structure of certain elliptic curves enables new capabilities that have far-reaching implications in cryptography. Specifically, we focus on elliptic curves that support a "pairing." As we will see, the pairing enables a broad range of new cryptographic capabilities beyond basic public-key cryptography. We begin with the basic definition:

Definition 2.1 (Symmetric Pairing Group)**.** A symmetric pairing group $(\mathbb{G}, \mathbb{G}_T, e)$ consists of two groups $\mathbb{G}$ (with generator g) and $\mathbb{G}_T$, each of prime order p, and an efficiently-computable mapping $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ (called the "pairing") that satisfies the following properties:

- **Non-triviality:** It holds that $e(g, g) \neq 1$. In other words, $e(g, g)$ is a generator of $\mathbb{G}_T$.
- **Bilinearity:** For all $x, y \in \mathbb{Z}_p$, it holds that $e(g^x, g^y) = e(g, g)^{xy}$.

We typically refer to $\mathbb{G}$ as the "base group" and $\mathbb{G}_T$ as the "target group."

Intuitively, the pairing provides an efficient way to "multiply in the exponent." Namely, if we think of g^x as an encoding of $x \in \mathbb{Z}_p$ and g^y as an encoding of $y \in \mathbb{Z}_p$, then $e(g^x, g^y)$ is an encoding of $xy \in \mathbb{Z}_p$ in $\mathbb{G}_T$ (with respect to the generator $e(g, g)$). An immediate consequence is that the DDH assumption does not hold in the base group $\mathbb{G}$. Specifically, we can construct the following distinguisher:

- On input a DDH challenge (g, u, v, w), output 1 if $e(u, v) = e(g, w)$ and 0 otherwise.

We claim that this breaks DDH with advantage close to 1:

- If $u = g^x$, $v = g^y$, and $w = g^{xy}$, then

$$e(u, v) = e(g^x, g^y) = e(g, g)^{xy} = e(g, g^{xy}) = e(g, w).$$

  In this case, the algorithm always outputs 1.

- If $u = g^x$, $v = g^y$, and $w = g^r$ where $r \xleftarrow{R} \mathbb{Z}_p$, then with probability $1 - 1/p$, it holds that $r \neq xy$. In this case,

$$e(u, v) = e(g^x, g^y) = e(g, g)^{xy} \neq e(g, g^r) = e(g, w).$$

  In this case, the algorithm outputs 0 with probability $1 - 1/p$.

Thus, the pairing yields a DDH distinguisher that succeeds with probability $1 - 1/p$. While the DDH assumption does not hold in the group $\mathbb{G}$, other problems such as the discrete log problem or the CDH problem can still plausibly hold in $\mathbb{G}$. Observe that if a particular assumption holds in $\mathbb{G}$, then it must also hold in $\mathbb{G}_T$ (since we can always lift an element in the base group to one in the target group). We will also introduce new computational assumptions that are conjectured to hold in pairing groups.

An application: tripartite Diffie-Hellman. We will begin with a simple application of pairings to three-party key-agreement introduced by Joux [Jou00]. Recall that the classic Diffie-Hellman key-agreement protocol allows two parties (e.g., Alice and Bob) to securely agree on a shared key (by broadcasting just a single message). Suppose instead we had three parties: Alice, Bob, and Carol. Can we construct a non-interactive key-agreement protocol? With pairing-free groups, it is not clear how to do this, but with pairings, this is straightforward. We will work in a (symmetric) pairing group $(\mathbb{G}, \mathbb{G}_T, e)$ of order p and generator g.

- We will label the three parties with an index $i \in \{1, 2, 3\}$.
- Each party starts by sampling an exponent $x_i \xleftarrow{R} \mathbb{Z}_p$ and publishes $g^{x_i} \in \mathbb{G}$. This is the exact same setup as standard Diffie-Hellman key exchange.
- The shared key is $e(g, g)^{x_1 x_2 x_3}$.

First, we observe that each party is able to compute the shared key (with the help of the pairing). Consider the view of Party 1:

- Party 1 chooses the exponent $x_1 \xleftarrow{R} \mathbb{Z}_p$ and receives g^{x_2} and g^{x_3} from the other two parties.
- Party 1 can compute the shared key using the pairing: $K = e(g^{x_2}, g^{x_3})^{x_1} = e(g, g)^{x_1 x_2 x_3}$. Formally, the actual key would be obtained by applying a randomness extractor (i.e., a hash function) to K.

Parties 2 and 3 can perform an analogous calculation to obtain the shared key K: namely they pair the public keys from the other two parties, and then exponentiates the result to their own secret key. The security of the protocol relies on the assumption that it should be difficult to compute (or distinguish) $K = e(g, g)^{x_1 x_2 x_3}$ given only $(g, g^{x_1}, g^{x_2}, g^{x_3})$. These assumptions can be viewed as the analog of the CDH and DDH assumptions in the pairing-based world. Both assumptions are defined with respect to a pairing group $(\mathbb{G}, \mathbb{G}_T, e)$ of order p and generator g:

- **Bilinear Diffie-Hellman (BDH) problem:** Given (g, g^x, g^y, g^z) where $x, y, z \xleftarrow{R} \mathbb{Z}_p$, compute $e(g, g)^{xyz} \in \mathbb{G}_T$.
- **Decisional bilinear Diffie-Hellman (DBDH) problem:** Distinguish $(g, g^x, g^y, g^z, e(g, g)^{xyz})$ from $(g, g^x, g^y, g^z, e(g, g)^r)$ where $x, y, z, r \xleftarrow{R} \mathbb{Z}_p$.

We say the BDH (resp., DBDH) assumption holds with respect to a pairing group $(\mathbb{G}, \mathbb{G}_T, e)$ if the advantage of every efficient adversary in solving the BDH problem (resp., DBDH) problem is negligible (resp., negligibly close to $1/2$).[1]

In traditional groups, it is easy to compute 'linear" functions in the exponent (using the group operation), while computing quadratic functions is believed to be difficult (i.e., this is the CDH problem). In a pairing group, it is easy to compute quadratic functions in the exponent (via the pairing), while computing *cubic* functions is believed to be difficult (i.e., this is the BDH problem). It turns out that the difference between being able to compute linear functions vs. being able to compute quadratic functions is very significant.

[1]Formally, the group description $(\mathbb{G}, \mathbb{G}_T, e)$ is sampled from a distribution that is parameterized by the security parameter. The adversary's running time and advantage are correspondingly parameterized by the security parameter.

Beyond three parties. As illustrated above, pairings immediately give a solution to the problem of three-party non-interactive key-agreement. A natural question is whether we can go beyond three parties. In general, can we come up with an n-party non-interactive key-agreement protocol? One possibility is to use an $(n-1)$-way *multilinear map* (i.e., an n-way multilinear map $e\colon \mathbb{G}^n \to \mathbb{G}_T$ is an efficient mapping where $e(g^{x_1}, \ldots, g^{x_n}) = e(g, \ldots, g)^{x_1 \cdots x_n}$). However, we do not have algebraic constructions of groups with n-linear maps (for $n > 2$) where the multilinear map is efficiently computable and the discrete log assumption still plausibly holds over the group. The only approach we have for constructing multilinear maps (and more generally, n-party key-agreement) is through the use of indistinguishability obfuscation. This is a powerful cryptographic primitive that can be built from standard assumptions, but requires very heavy machinery to realize. It remains an open question to come up with a simple (and concretely-efficient) n-party key-agreement protocol (or more generally, construct an efficient n-way multilinear map over a group where the discrete log problem plausibly holds).

Where do pairings come from? As noted earlier, pairing groups are built from elliptic curve groups. There are two main types of pairing groups: (1) a symmetric pairing group (Definition 2.1) where there is a single base group $\mathbb{G}$ and the pairing map is a mapping $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$; and (2) an asymmetric pairing group where there are *two* base groups $\mathbb{G}_1, \mathbb{G}_2$ and the pairing is an efficient mapping $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. In the asymmetric setting, the groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ are all cyclic groups of the same order. Bilinearity in the asymmetric setting says the following:

$$\forall x, y \in \mathbb{Z} : e(g_1^x, g_2^y) = e(g_1, g_2)^{xy},$$

where $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ are generators of their respective groups. As in the symmetric case, we require that $e(g_1, g_2)$ be a generator of the target group $\mathbb{G}_T$. Most pairing-based constructions can be adapted to work with both symmetric pairing groups or asymmetric pairing groups. In this course, we will see examples in both settings, depending on which instantiation is simpler to describe.

The pairing groups we use in cryptography are based on elliptic curve groups. Specifically, let $E(\mathbb{F}_p)$ be the group of points on an elliptic curve E with coefficients in $\mathbb{F}_p$. In the case of an asymmetric pairing group of prime order q, the groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ are typically instantiated as follows:[2]

- The base group $\mathbb{G}_1 \subseteq E(\mathbb{F}_p)$ is an order-q subgroup of $E(\mathbb{F}_p)$.
- The base group $\mathbb{G}_2 \subseteq E(\mathbb{F}_{p^d})$ is an order-q subgroup of the elliptic curve group defined over the *extension* field $\mathbb{F}_{p^d}$ where $d > 1$.[3] Moreover, $\mathbb{G}_2 \cap \mathbb{G}_1 = \{\mathcal{O}\}$.
- The target group $\mathbb{G}_T \subseteq \mathbb{F}_{p^d}^\times$ is a multiplicative subgroup of the extension field $\mathbb{F}_{p^d}$ of order q.

In other words, the pairing takes an element over the elliptic curve over $\mathbb{F}_p$, an element on the elliptic curve defined over the extension $\mathbb{F}_{p^d}$, and outputs an element in the extension field itself. The constant d is a property of the elliptic curve and is called the *embedding degree* of the elliptic curve E (the embedding degree d is the smallest value where $\mathbb{F}_{p^d}$ contains a multiplicative subgroup of order q, or equivalently, the smallest d such that $q \mid p^d - 1$). Since the base group $\mathbb{G}_2$ and $\mathbb{G}_T$ are defined over the extension field $\mathbb{F}_{p^d}$,

[2] Symmetric pairings groups can be constructed in a similar manner, albeit over "supersingular" elliptic curves. For simplicity, we will just describe the instantiation for asymmetric pairing groups here.

[3] We will not formally define extension fields in this class, but concretely, the elements of $\mathbb{F}_{p^d}$ can be viewed as polynomials of degree $d-1$ with coefficients in $\mathbb{F}_p$. The elliptic curve group $E(\mathbb{F}_{p^d})$ is the set of points $(x, y) \in \mathbb{F}_{p^d} \times \mathbb{F}_{p^d}$ that satisfy the curve equation E. Note that $\mathbb{F}_{p^d}$ contains $\mathbb{F}_p$ as a sub-field, we always have that $E(\mathbb{F}_p) \subseteq E(\mathbb{F}_{p^d})$.

the embedding degree has to be small for the pairing to be efficiently-computable. Thus, elliptic curves with a small embedding degree (e.g., $d \leq 16$) are said to be "pairing-friendly." Standard elliptic curves like P-256 or `Curve25519` are *not* pairing-friendly. For instance, the embedding degree of P-256 is $d > 2^{253}$ and that of `Curve25519` is $d > 2^{249}$. As we discuss more below, having a high embedding degree is desirable for security when considering *pairing-free* elliptic-curve groups.

Pairings in practice. A common pairing-friendly elliptic curve group is `bls381`, which is a Barreto-Lynn-Scott (BLS) curve [BLS02] and conjectured to provide roughly 128-bits of security. The `bls381` curve has embedding degree $d = 12$ and is of prime order q where q is a 256-bit prime. As the name suggests, the elliptic curve itself is defined over $\mathbb{F}_p$ where p is a 381-bit prime. We now discuss some practical considerations:

- Elements of $\mathbb{G}_1$ can be described by a pair of points $(x, y) \in \mathbb{F}_p$. Note that given the value of $x \in \mathbb{F}_p$ and the curve equation E, the value of y is fully determined, up to its sign. Thus, we can define the compressed representation of an elliptic curve point (x, y) to be x along with a sign bit; this trick is often referred to as "point compression." The size of an element in $\mathbb{G}_1$ is thus $381/8 \approx 48$ bytes.
- Naïvely, an element of $\mathbb{G}_2$ would be a pair of points $(x, y) \in \mathbb{F}_{p^d}$. Since $d = 12$, this would normally require storing at least 12 elements of $\mathbb{F}_p$, which will be expensive for computation and for communication or storage. However, there is a clever way to reduce these costs. The trick is to rely on the fact that there is an efficiently-computable injective group homomorphism ψ from $E'(\mathbb{F}_{p^2})$ to $E(\mathbb{F}_{p^{12}})$, where E' is an elliptic curve related to E (specifically, E' is the "twist" of E). Thus, instead of working over $E(\mathbb{F}_{p^{12}})$, we instead treat $\mathbb{G}_2 = E'(\mathbb{F}_{p^2})$. When we need to apply the pairing, then we first apply ψ to the element in $\mathbb{G}_2$ to obtain an element over $E(\mathbb{F}_{p^{12}})$, which we then use in the pairing. Observe now that elements of $\mathbb{G}_2$ can be represented by a pair of elements in $\mathbb{F}_p$ (as opposed to $d = 12$ elements). Correspondingly, group operations in $\mathbb{G}_2$ are roughly $2\times$ slower than those in $\mathbb{G}_1$. The size of an element in $\mathbb{G}_2$ is twice that of an element in $\mathbb{G}_1$, or 96 bytes.
- The target group is the finite field $\mathbb{F}_{p^{12}}$, so each element is 576 bytes.

In terms of computational costs, the pairing is roughly $10\times$ slower than exponentiation in the base group $\mathbb{G}_1$.

Using pairings to solve discrete log. Recall from above that the pairing is a mapping $e\colon E(\mathbb{F}_p) \times E(\mathbb{F}_{p^d}) \to \mathbb{F}_{p^d}$. Notably, the output of the pairing is *not* a point on an elliptic curve. Instead, it is an element of the finite field $\mathbb{F}_{p^d}$. As mentioned before, there are *sub-exponential* time algorithms for solving discrete log over finite fields (based on index calculus). One of the first application of pairings to cryptography was proposed by Menezes, Okamoto, and Vanstone as an algorithm for solving the discrete log problem over an elliptic curve group [MVO91].

Specifically, if E is an elliptic curve with low embedding degree $d \in \mathbb{N}$ over $\mathbb{F}_p$, then the discrete log problem over $E(\mathbb{F}_p)$ is no harder than the discrete log problem over $\mathbb{F}_{p^d}$. This follows by the following observation: given $h = g_1^x$ where $g \in E(\mathbb{F}_p)$, the algorithm uses the pairing to project $h = g^x$ onto $e(h, g_2) = e(g_1, g_2)^x \in \mathbb{F}_{p^d}$. Recovering $x \in \mathbb{Z}_q$ is now the problem of computing the discrete log of $e(h, g_2)$ base $e(g_1, g_2)$ in $\mathbb{F}_{p^d}$. This is a discrete log problem over $\mathbb{F}_{p^d}$, which can be tackled using index calculus algorithm in sub-exponential time (i.e., sub-exponential in the order p^d). The MOV attack is the reason we prefer elliptic curves with high embedding degree. Of course, if we want to also use the pairing *constructively*, then we need pairing-friendly curves with an embedding degree d that is small enough to be efficient, but large enough such that the discrete log problem in $\mathbb{F}_{p^d}$ is still sufficiently hard (e.g., for 128-bits of security, we *minimally* require that $p^d \geq 2048$).

CS 395T: Topics in Cryptography January 23, 2024

Lecture 3: Short Signatures

Lecturer: David Wu *Scribe: Alex Burton*

In the previous lecture, we saw how pairings can be used to break problems such as DDH. In this lecture, we will see how to build new cryptography from pairings. The key intuition behind our first construction is that a pairing allows us to easily solve DDH (since we can check for equality in $\mathbb{G}_T$), but is not directly useful in solving CDH (since we cannot go from $\mathbb{G}_T$ back into $\mathbb{G}$). This idea is the basis for the BLS digital signature scheme.

Digital signature schemes. First, we recall several signature schemes and compare their signature lengths relative to concrete and asymptotic security parameters (where the bounds are based on the best-known attack on the associated assumptions).

Scheme	128 bits of security	λ bits of security
RSA	3072 bit signatures	$\tilde{O}(\lambda^3)$ bit signatures
ECDSA	512 bit signatures	4λ bit signatures
Schnorr	384 bit signatures	3λ bit signatures
BLS	256 bit signatures	2λ bit signatures
Obfuscation-Based	128 bit signatures	λ bit signatures

The table contains the Boneh-Lynn-Shacham (BLS) signature scheme [BLS01],[1] which we are about to introduce. Certainly, to achieve λ-bit of security, the signature must be at least λ-bits long. Thus, the obfuscation-based construction [SW14] is essentially *optimal* in terms of signature size. However, this scheme is currently only of theoretical interest. The concrete costs of signature verification for the obfuscation-based scheme is too large to be practically viable (or even implementable). Thus, BLS signatures give the shortest signatures that can be used in practical applications.

PRFs from DDH. The starting point for the BLS signature scheme is the following DDH-based PRF (in the random oracle model). This construction uses a random oracle, which we will define precisely later. Informally, one can think of a random oracle as a random function that everyone has access to.

Construction 3.1 (PRF from DDH in the Random Oracle Model)**.** Let $\mathbb{G}$ be a DDH-hard group of order p. Let $\mathcal{H}\colon \{0,1\}^n \to \mathbb{G}$ be a random oracle. Define a PRF as follows:

- $\mathsf{Setup}(1^\lambda)$: Sample and output $k \xleftarrow{R} \mathbb{Z}_p$.
- $\mathsf{Eval}(k, x)$: Output $\mathcal{H}(x)^k$.

To argue security informally: consider an adversary that evaluates the PRF on inputs $x_1, \ldots, x_q$. Then, it learns the pairs $(y_1, y_1^k), \ldots, (y_q, y_q^k)$ where $y_i = \mathcal{H}(x_i)$. Since $\mathcal{H}$ is a random oracle, the distribution of each y_i is independent and uniform over $\mathbb{G}$. A hybrid argument using DDH allows us to replace each y_i^k with $z_i \xleftarrow{R} \mathbb{G}$ one by one.[2] Thus, every output of the PRF appears random.

[1] The "BLS signature scheme" [BLS01] is a different BLS than that of "BLS curves." [BLS02]

[2] For this hybrid, it is convenient to use the following assumption equivalent to DDH: the distribution $\{(g, g^s, h, h^s) : g, h \xleftarrow{R} \mathbb{G}, s \xleftarrow{R} \mathbb{Z}_p\}$ is computationally indistinguishable from $\{(g, g^s, h, z) : g, h, z \xleftarrow{R} \mathbb{G}, s \xleftarrow{R} \mathbb{Z}_p\}$. Note that we can also get a *tight* reduction (without a hybrid argument) by relying on random self-reducibility of DDH.

Oblivious PRFs and oblivious transfer. The PRF above can be extended into a primitive known as an oblivious PRF (OPRF). In this scenario, the server holds a PRF secret key k, and a client wants to learn $\mathsf{Eval}(k, x)$ without revealing x to the server. To build this protocol from the previous PRF construction, the client samples $r \xleftarrow{R} \mathbb{Z}_p$ and sends $y = \mathcal{H}(x)^r$ to the server. The server replies with $z = y^k = \mathcal{H}(x)^{rk}$, from which the client computes $z^{r^{-1} \bmod p} = \mathcal{H}(x)^k$. Observe that the client's message y completely hides x *unless* $\mathcal{H}(x) = 1$, which for any choice of x, happens with $1/p$ probability over the oracle's randomness.

Note that there is a subtlety if we want to establish the following security property of the above OPRF: the client ought to only learn m many PRF evaluations from m executions of the OPRF evaluation. It turns out that this requires a stronger assumption than DDH: observe that a client can learn $y_d := g^{k^d}$ by first setting $y_0 := g$. Then, for each $i \in [d]$, the client passes y_{i-1} to the server, who replies with $y_i := y_{i-1}^k = g^{k^i}$. This allows the client to learn $g, g^k, g^{k^2}, \ldots, g^{k^d}$. It turns out that computing the discrete log k given the sequence $(g, g^k, g^{k^2}, \ldots, g^{k^d})$ is easier than computing the discrete log over $\mathbb{G}$. Specifically, if $d \mid p-1$, then there is a discrete log algorithm that recovers k in time $\tilde{O}(\sqrt{p/d} + \sqrt{d})$. A generic discrete log algorithm over $\mathbb{G}$ would run in time $\tilde{O}(\sqrt{p})$. We leave it as a homework exercise to develop this better discrete log algorithm.

OPRFs are useful since they immediately imply oblivious transfer (OT). In an OT protocol, a sender has two messages m_0, m_1, and a receiver wants to learn the message m_b where $b \in \{0, 1\}$. The privacy requirement is that the sender does not learn the selection bit b, and the receiver does not learn the unqueried message m_{1-b}. It turns out that oblivious transfer is both *necessary* and *sufficient* for general multiparty computation (i.e., a protocol that allows a group of mutually distrusting parties to compute an arbitrary function over their joint *private* inputs).

To obtain an OT protocol from an OPRF scheme, we proceed as follows. The sender starts by sending the blinded messages $c_0 = m_0 \oplus \mathsf{Eval}(k, 0)$ and $c_1 = m_1 \oplus \mathsf{Eval}(k, 1)$.[3] The receiver then executes an OPRF evaluation to learn $\mathsf{Eval}(k, b)$, from which they can recover $m_b = c_b \oplus \mathsf{Eval}(k, b)$. This is an example of a 1-out-of-2 OT protocol (i.e., the sender has two messages and the receiver is obliviously selecting one of them). We can also consider a 1-out-of-N protocol where the sender has N messages $m_1, \ldots, m_N$ and the receiver's goal is to learn m_i without revealing its index $i \in [N]$. The basic OPRF protocol directly generalizes to this setting.

BLS signatures. We now introduce BLS digital signatures [BLS01]. One way to view this construction is to apply the standard "MAC from PRF" construction to Construction 3.1, and replace the secret key verification procedure with a public key verification procedure via the pairing.

Construction 3.2 (Boneh-Lynn-Shacham Digital Signatures [BLS01])**.** Let $(\mathbb{G}, \mathbb{G}_T, e)$ be a CDH-hard symmetric pairing group with generator g and prime order p. Let $\mathcal{H}\colon \{0,1\}^n \to \mathbb{G}$ be a hash function (modeled as a random oracle). Define a digital signature scheme as follows:

- $\mathsf{KeyGen}(1^\lambda)$: Sample $k \xleftarrow{R} \mathbb{Z}_p$. Output the signing key $\mathsf{sk} = k$ and the verification key $\mathsf{vk} = g^k$.
- $\mathsf{Sign}(\mathsf{sk}, m)$: Output the signature $\sigma = \mathcal{H}(m)^{\mathsf{sk}}$.
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: Check if $e(\mathsf{vk}, \mathcal{H}(m)) = e(g, \sigma)$.

Correctness is immediate from properties of e:

$$e(g^k, \mathcal{H}(m)) = e(g, \mathcal{H}(m))^k = e(g, \mathcal{H}(m)^k).$$

[3] Here, assume that the output space of the PRF is a bit-string. This can be assumed without loss of generality by composing the PRF with a randomness extractor.

Random oracles. Before we give the security proof for BLS, we recall the precise notion of a random oracle and the random oracle model. A *random oracle* (RO) $\mathcal{H}$ is simply an oracle to a random function. One can view queries to a random oracle like a "remote procedural call;" notably, such a query is distinct from a circuit or piece of code that can be evaluated. The *random oracle model* (ROM) is a model in which everyone has public access to a random oracle. Importantly, the random oracle model is *not* an assumption. In practice, SHA-256 is commonly used to *heuristically* instantiate the random oracle.

The power of the ROM is that a reduction algorithm gets to respond to an adversary's queries to the random oracle, provided that the responses have the same distribution as a random function. This technique is called "programming the random oracle."

The paradox of secure signatures. There is an apparent paradox in proving the security of a digital signature scheme from a concrete assumption. Consider an adversary $\mathcal{A}$ and a reduction algorithm $\mathcal{B}$. On the one hand, $\mathcal{B}$ must be able to respond to signing queries made by $\mathcal{A}$. However, $\mathcal{B}$ *cannot* know how to sign the message for which $\mathcal{A}$ produces a forgery—otherwise, $\mathcal{B}$ could replace $\mathcal{A}$'s forgery with a signature computed by itself, but then $\mathcal{B}$ breaks the assumption unconditionally!

Programming the random oracle allows us to sidestep this paradox. Let m^* be the message on which $\mathcal{A}$ forges. In essence, we will program $\mathcal{H}(m) := g^{\alpha_m}$ for $m \neq m^*$ where the reduction algorithm chooses $\alpha_m \xleftarrow{R} \mathbb{Z}_p$ itself. The reduction algorithm then defines $\mathcal{H}(m^*)$ to be an embedding of the CDH challenge. In particular, the reduction algorithm will be able sign any message *except for* m^*, and $\mathcal{A}$'s forgery on m^* will allow us to solve CDH. Of course, we don't know m^* ahead of time; we resolve this by guessing.

Theorem 3.3 (Security of BLS signatures)**.** *If CDH holds in $\mathbb{G}$, then the BLS scheme is secure (i.e., existentially unforgeable under a chosen-message attack) in the random oracle model.*

Proof. Suppose $\mathcal{A}$ is an adversary for the signature security game. Assume without loss of generality that $\mathcal{A}$ has the following properties:[4]

(a) $\mathcal{A}$ makes at most Q queries to the random oracle for some polynomial $Q = \mathsf{poly}(\lambda)$.

(b) $\mathcal{A}$ always queries the random oracle for m before querying for a signature on m.

(c) $\mathcal{A}$ always queries the random oracle for m^*, where (m^*, σ^*) is the forgery attempt.

(d) For a given message m, $\mathcal{A}$ make at most one random oracle query for m.

Define the adversary $\mathcal{B}$ for CDH as follows.

- Let $(g, u = g^x, v = g^y)$ be the CDH challenge. Run $\mathcal{A}$, and give $\mathsf{vk} = u$ to $\mathcal{A}$.
- Guess an index $i^* \xleftarrow{R} [Q]$.
- Given the i^{th} random oracle query from $\mathcal{A}$, for the message m:
  - If $i = i^*$, reply with $\mathcal{H}(m) := v$.
  - Otherwise, sample $\alpha_m \xleftarrow{R} \mathbb{Z}_p$ and reply with $\mathcal{H}(m) := g^{\alpha_m}$.
- Given a signing query from $\mathcal{A}$ for the message m:

[4] We can justify these as follows. (a) is immediate, since $\mathcal{A}$ is efficient. For (b) resp. (c): given any $\mathcal{A}$, we can easily construct $\mathcal{A}'$ (with identical advantage to $\mathcal{A}$) that satisfies (b) resp. (c) by adding random oracle queries and discarding the result. For (d): we can construct $\mathcal{A}'$ that maintains a table of previous random oracle queries, and uses that table for any repeated query.

- If m was the $(i^*)^{\text{th}}$ query to the random oracle, abort with output $\perp$.
- Otherwise, reply with $\sigma_m := u^{\alpha_m}$.

- Let (m^*, σ^*) be the output from $\mathcal{A}$. Output σ^*.

We now claim that

$$\mathsf{CDHAdv}[\mathcal{B}] \geq \frac{1}{Q}\mathsf{SigAdv}[\mathcal{A}].$$

Consider the event E that i^* is the correct guess (i.e., the forged message m^* was the $(i^*)^{\text{th}}$ query to the random oracle. Event E occurs with probability $1/Q$ and independently of $\mathcal{A}$'s execution.[5]

Now conditioning on E, $\mathcal{A}$'s view of the random oracle is correct (uniformly random), since in either case the output is g^r for a random $r \xleftarrow{R} \mathbb{Z}_p$. $\mathcal{A}$'s view of the signing queries is also correct: the first case cannot occur, since $\mathcal{A}$ does not makes a signing query on m^*, and in the second case, $u^{\alpha_m} = g^{\alpha_m x} = \mathcal{H}(m)^x = \mathsf{Sign}(x, m)$. Finally, if σ^* is a valid forgery, then

$$e(u, \mathcal{H}(m^*)) = e(g, \sigma^*) \implies e(g, g)^{xy} = e(g, \sigma^*) \implies \sigma^* = g^{xy}.$$

So σ^* is indeed the CDH solution. □

Tight reductions for BLS signatures. The factor $1/Q$ in the advantage analysis is called the *security loss*. This factor is impactful in practice: for example, if we want to support $Q \sim 2^{20}$ queries for a 128-bit secure signature, we would need a 148-bit hard pairing group.

It turns out that there is a straightforward adaptation to BLS that allows for a tight reduction (i.e., a constant security loss of $1/2$ instead of $1/Q$). This is left as an exercise for the reader. As a hint, the modified construction will have two valid signatures for each message, and the signatures will be one bit longer.

[5] There is a subtlety here that we are glossing over: it is not immediately clear that $\mathcal{A}$'s view is actually independent of i^*. One way to fix this is to first consider an intermediate (hybrid) experiment that is identical to the normal game except the *challenger* randomly chooses $i^* \xleftarrow{R} [Q]$ at the beginning; in addition to the normal win condition, the adversary's forgery message m^* must be the $(i^*)^{\text{th}}$ random oracle query. Clearly, going from the normal game to this intermediate experiment reduces the win probability of any adversary by exactly a factor of $1/Q$, since i^* is actually independent of $\mathcal{A}$'s view. Once we are in the intermediate experiment, our reduction algorithm $\mathcal{B}$ gets to set i^* itself.

CS 395T: Topics in Cryptography January 25, 2024

Lecture 4: Aggregate and Threshold Signatures

Lecturer: David Wu *Scribe: Alex Burton*

BLS signatures, introduced in the previous lecture, have many useful properties. We start by showing how to construct aggregate signatures—a scheme in which signatures can be combined into a single short signature—essentially for free [BGLS03].

Definition 4.1 (Aggregate Signature Scheme)**.** An aggregate signature scheme is a digital signature scheme augmented with a pair of efficient algorithms $(\mathsf{Aggregate}, \mathsf{VerifyAggregate})$:

- $\mathsf{Aggregate}((\mathsf{vk}_1, m_1, \sigma_1), \ldots, (\mathsf{vk}_n, m_n, \sigma_n)) \to \sigma_{\mathsf{agg}}$: Given a set of verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$, messages $m_1, \ldots, m_n$, and associated signatures $\sigma_1, \ldots, \sigma_n$, the aggregate algorithm outputs a single aggregated signature σ_{agg}.
- $\mathsf{VerifyAggregate}((\mathsf{vk}_1, \ldots, \mathsf{vk}_n), (m_1, \ldots, m_n), \sigma_{\mathsf{agg}}) \to \{0, 1\}$: Check if σ_{agg} is a valid signature for the messages $(m_1, \ldots, m_n)$ with respect to verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$.

Correctness. The correctness requirement is the following: if $\mathsf{Verify}(\mathsf{vk}_i, m_i, \sigma_i) = 1$ for all $i \in [n]$ and $\sigma_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}((\mathsf{vk}_1, m_1, \sigma_1), \ldots, (\mathsf{vk}_n, m_n, \sigma_n))$. Then

$$\Pr[\mathsf{VerifyAggregate}((\mathsf{vk}_1, \ldots, \mathsf{vk}_n), (m_1, \ldots, m_n), \sigma_{\mathsf{agg}}) = 1] = 1.$$

Succinctness. A trivial aggregate signature scheme is to take $\sigma_{\mathsf{agg}} = (\sigma_1, \ldots, \sigma_n)$. Since the whole point of aggregation is to reduce communication, we generally only consider aggregate signatures where $|\sigma_{\mathsf{agg}}| = \mathsf{poly}(\lambda, \log n)$. Technically, an aggregate signature with size sublinear in n (e.g., $|\sigma_{\mathsf{agg}}| = o(n) \cdot \mathsf{poly}(\lambda)$) is also interesting.

Applications and multi-signatures. Aggregate signatures reduce the communication cost of having to send multiple signatures at once. A simple application is aggregating the certificate chain in TLS session setup; another is aggregating signatures in blockchain (e.g., aggregating the signatures on different transactions within a block). Another example is a setting where many authorities need to sign off on a certain message, (e.g., issuing a nuclear launch command). In this setting, the aggregation is performed over signatures on the *same* message. This is often referred to as a *multi-signature*.

Security. Defining security for signature aggregation is more subtle. We first describe an attack on a "natural" aggregate version of BLS, which is to simply multiple the input signatures together:

- $\mathsf{Aggregate}((\mathsf{vk}_1, m_1, \sigma_1), \ldots, (\mathsf{vk}_n, m_n, \sigma_n)) \to \sigma_{\mathsf{agg}}$: Output $\sigma_{\mathsf{agg}} \leftarrow \prod_{i \in [n]} \sigma_i$.
- $\mathsf{VerifyAggregate}((\mathsf{vk}_1, \ldots, \mathsf{vk}_n), (m_1, \ldots, m_n), \sigma_{\mathsf{agg}})$: Check that $e(g, \sigma_{\mathsf{agg}}) \stackrel{?}{=} \prod_{i \in [n]} e(\mathsf{vk}_i, \mathcal{H}(m_i))$.

Correctness is satisfied, since

$$e(g, \sigma_{\mathsf{agg}}) = e\left(g, \prod_{i \in [n]} \sigma_i\right) = \prod_{i \in [n]} e\left(g, \sigma_i\right) = \prod_{i \in [n]} e(\mathsf{vk}, \mathcal{H}(m_i)).$$

Remark 4.2 (Fast Verification for Multi-Signatures)**.** As described above, signature verification requires computing n pairings. For the particular case of a multi-signature (i.e., where $m_i = m$ for all $i \in [n]$), we can first "aggregate" the verification keys and implement the verification algorithm with a *single* pairing. This follows by the following calculation:

$$\prod_{i\in[n]} e(\mathsf{vk}_i, \mathcal{H}(m)) = e\left(\prod_{i\in[n]} \mathsf{vk}_i, \mathcal{H}(m)\right) = e(\mathsf{vk}_{\mathsf{agg}}, \mathcal{H}(m)),$$

where $\mathsf{vk}_{\mathsf{agg}} = \prod_{i\in[n]} \mathsf{vk}_i$ is the "aggregated" verification key. Namely, the aggregate verification algorithm now checks $e(g, \sigma_{\mathsf{agg}}) \stackrel{?}{=} e(\mathsf{vk}_{\mathsf{agg}}, \mathcal{H}(m))$, which requires a single pairing operation. For settings where one frequently verifies signatures from the same quorum of users (i.e., the same set of vk_i), one can precompute and cache the aggregated verification key $\mathsf{vk}_{\mathsf{agg}}$. In this case, verifying a multi-signature has the same cost as verifying a single signature.

Rogue key attacks. However, the basic scheme described above is trivially broken in the following sense. Suppose Alice's verification key is $\mathsf{vk}_A = g^{k_A}$. An attacker can come up with a "bad" verification key vk' such that it is easy to forge an aggregate signature with respect to $(\mathsf{vk}_A, \mathsf{vk}')$. In particular, the attacker can choose $x \xleftarrow{R} \mathbb{Z}_p$ and set $\mathsf{vk}' = g^x \cdot \mathsf{vk}_A^{-1} = g^{x-k_A}$. Then $\sigma^*_{\mathsf{agg}} = \mathcal{H}(m^*)^x$ satisfies $\mathsf{VerifyAggregate}((\mathsf{vk}_A, \mathsf{vk}'), (m^*, m^*), \sigma^*_{\mathsf{agg}}) = 1$:

$$\begin{aligned} e(g, \sigma^*_{\mathsf{agg}}) &= e(g, \mathcal{H}(m^*))^x = e(g, \mathcal{H}(m^*))^{k_A} \cdot e(g, \mathcal{H}(m^*))^{x-k_A} \\ &= e(\mathsf{vk}_A, \mathcal{H}(m^*)) \cdot e(\mathsf{vk}', \mathcal{H}(m^*)). \end{aligned}$$

We refer this as a "rogue key attack" since the adversary can register a new public key (correlated with an honest user's public key) and in turn, forge an aggregate signature on behalf of the honest user. Thus, for an aggregate signature scheme to be secure, the adversary should not be able to claim an honest user signed a message they did not explicitly sign. We capture this in the following definition:

Definition 4.3 (Secure Aggregation)**.** Consider the following security game between an adversary and a challenger:

- The challenger samples $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. The challenger sends vk to the adversary.
- The challenger allows the adversary to make arbitrarily many signing queries $\mathsf{Sign}(\mathsf{sk}, \cdot)$.
- The adversary outputs a forgery $((\mathsf{vk}^*_1, \ldots, \mathsf{vk}^*_n), (m^*_1, \ldots, m^*_n), \sigma^*_{\mathsf{agg}})$.

The adversary wins if the following conditions hold:

- $\mathsf{VerifyAggregate}((\mathsf{vk}^*_1, \ldots, \mathsf{vk}^*_n), (m^*_1, \ldots, m^*_n), \sigma^*_{\mathsf{agg}}) = 1$,
- For some $i \in [n]$, $\mathsf{vk}^*_i = \mathsf{vk}$ and the adversary did not query for a signature on m_i.

An aggregate signature scheme is secure if no efficient adversary $\mathcal{A}$ in the above game can win with non-negligible probability.

Securely aggregating BLS signatures. We describe two approaches to modify BLS signatures to support secure aggregation:

1. **Sign the verification key in addition to the message:** Instead of signing m, the user now signs the pair (vk, m). In other words, the signature is now $\mathcal{H}(\mathsf{vk}\|m)^k$. Correspondingly, the aggregate verification relation is now
$$e(g, \sigma_{\mathsf{agg}}) \stackrel{?}{=} \prod_{i \in [n]} e(\mathsf{vk}_i, \mathcal{H}(\mathsf{vk}_i, m_i)).$$
2. **Add a proof of possession for the verification key:** In the case of BLS, the proof can be implemented by a signature on the verification key. In particular, the verification key is now $\mathsf{vk} = (u, \pi)$ where $u = g^k$ and $\pi = \mathcal{H}'(u)^k$. The verification relation will check the validity of the verification key: namely, that $e(g, \pi) = e(u, \mathcal{H}'(u))$. Note that we need to use a different (and independent) random oracle $\mathcal{H}'$ or the scheme is insecure.

Threshold signatures. In threshold cryptography, the goal is to protect a cryptographic key by splitting it into many different pieces (i.e., "shares"). A compromise of a small number of shares should not compromise the overall scheme. For example, a certificate authority may distribute their signing key across many different machines such that even if a few machines get compromised, its root signing key remains secure. In this setting, the holder of the signing key sk can split it into n individual keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ and give each key-share to a different party. The i^{th} party can use sk_i to independently produce a partial signature σ_i on a message m. In a t-out-of-n scheme, a user should be able to take any subset of t partial signatures and combine them into a signature σ on m (with respect to the verification key vk associated with sk). The security requirement is that an adversary cannot forge a signature on any message m given $t-1$ partial signatures (or even $t-1$ shares of sk).

n-out-of-n threshold signatures for BLS. We start with an n-ouf-of-n threshold signature scheme. Given a BLS signing key $k \in \mathbb{Z}_p$, we sample $k_1, \ldots, k_{n-1} \xleftarrow{R} \mathbb{Z}_p$ and set $k_n \leftarrow k - \sum_{i \in [n-1]} k_i$. Equivalently, we are sampling $k_1, \ldots, k_n \xleftarrow{R} \mathbb{Z}_p$ conditioned on $\sum_{i \in [n]} k_i = k$. A partial signature σ_i on message m with respect to the i^{th} key-share k_i is a standard BLS signature on m under k_i: $\sigma_i = H(m)^{k_i}$. Given n signature shares $\sigma_1, \ldots, \sigma_n$ on the same message m, we can recover a signature on m by computing
$$\sigma = \prod_{i \in [n]} \sigma_i = \mathcal{H}(m)^{\sum_{i \in [n]} k_i} = \mathcal{H}(m)^k.$$
Intuitively, security holds because if a single share is unknown, then k is uniformly distributed in $\mathbb{Z}_p$.

Additive secret sharing. This construction implicitly uses a n-out-of-n threshold secret sharing scheme. In this setting, a secret $s \in \mathbb{Z}_p$ is split into many shares $s_1, \ldots, s_n$ such that any set of $n-1$ shares *perfectly* hide the secret s. In the scheme above, we secret share the key k into random shares $k_1, \ldots, k_n$ subject to $\sum_{i \in [n]} k_i = k$. We often refer to this as an *additive* secret sharing of the secret key k. Since the reconstruction algorithm (i.e., the algorithm to recover k from the shares $k_1, \ldots, k_n$ is linear—in fact, just taking the sum), we can implement it "in the exponent." This yields the n-out-of-n threshold version of BLS signatures described above.

In the following lecture, we will develop a more powerful approach for t-ouf-of-n secret sharing for *arbitrary* thresholds t. This construction (called Shamir secret sharing) also has a linear reconstruction algorithm and will allow us to construct a t-out-of-n threshold signature scheme.

CS 395T: Topics in Cryptography January 30, 2024

Lecture 5: Threshold Signatures

Lecturer: David Wu *Scribe: Jonathan Browne*

In the last lecture, we constructed n-out-of-n threshold signatures from BLS. Let $\mathsf{vk} = g^k$ be a BLS verification key and k be the associated signing key. We previously showed how to split the signing key k into n different shares $k_1, \ldots, k_n$ that can be distributed to n different parties. Each party can use their individual key share k_i to sign a message m; this yields a "signature share" σ_i on m (in fact, $\sigma_i = H(m)^{k_i}$ is simply a BLS signature on m under the i^{th} user's signing key). Given n signature shares $\sigma_1, \ldots, \sigma_n$ on a message m, a user can reconstruct a signature on the message m that verifies under the verification key vk. In this lecture, we will expand this notion to threshold BLS signatures where instead of requiring a signature share from *all* n parties, it suffices to obtain shares from any subset of t parties for some threshold t where $1 \leq t \leq n$. To do so, we will rely on properties of polynomials.

The polynomial interpolation problem. Let $\mathbb{F}$ be a field. Pick d distinct points $x_1, \ldots, x_d \in \mathbb{F}$. We will show that for every $y_1, \ldots, y_d \in \mathbb{F}$, there exists a *unique* polynomial $f : \mathbb{F} \to \mathbb{F}$ of degree at most $d-1$ such that for all $i \in [d]$, $f(x_i) = y_i$. Given $(x_1, y_1), \ldots, (x_d, y_d)$, the problem of finding such an f is known as the polynomial interpolation problem.

The Vandermonde matrix. Let $f(x) = \sum_{i \in [d]} f_{i-1} x^{i-1}$. Observe that we can express the evaluation of f at a point $x \in \mathbb{F}$ as an inner product between the coefficient vector $\mathbf{f} = [f_0 \mid f_1 \mid \cdots \mid f_{d-1}]^\intercal$ and the vector $\mathbf{x} = [1 \mid x \mid x^2 \mid \cdots \mid x^{d-1}]^\intercal$. In particular, we can write

$$f(x) = \mathbf{x}^\intercal \mathbf{f} = [x^0 \mid x^1 \mid \cdots \mid x^{d-1}] \cdot \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{d-1} \end{bmatrix} \in \mathbb{F}.$$

This naturally extends to evaluating the polynomial f at d points $x_1, \ldots, x_d$ as follows:

$$\underbrace{\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{d-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{d-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_d & x_d^2 & \cdots & x_d^{d-1} \end{bmatrix}}_{\mathbf{V}} \cdot \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{d-1} \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_d) \end{bmatrix} \tag{5.1}$$

We refer to the matrix $\mathbf{V} \in \mathbb{F}^{d \times d}$ as the *Vandermonde matrix* associated with the points $(x_1, \ldots, x_d)$. We now show that if $x_1, \ldots, x_d$ are distinct, then the Vandermonde matrix is invertible. Our analysis will rely on the fact that a univariate polynomial of degree d over a field $\mathbb{F}$ has at most d roots. This basic fact about polynomials has a remarkable number of applications (not only to cryptography but also to theoretical computer science).

Fact 5.1 (Roots of Polynomials)**.** Let $\mathbb{F}$ be a field and $f : \mathbb{F} \to \mathbb{F}$ be a non-zero (univariate) polynomial of degree at most d. Namely, we can write $f(x) = \sum_{i \in [d+1]} f_{i-1} x^{i-1}$ for some set of coefficients $f_0, \ldots, f_d \in \mathbb{F}$. Then, the polynomial f has at most d roots (i.e., there are at most d points $x_1, \ldots, x_d \in \mathbb{F}$ where $f(x_i) = 0$).

Lemma 5.2 (Invertibility of the Vandermonde Matrix)**.** *Let $x_1, \ldots, x_d \in \mathbb{F}$ be distinct points over a field $\mathbb{F}$, and let $\mathbf{V} \in \mathbb{F}^{d \times d}$ be the Vandermonde matrix associated with $(x_1, \ldots, x_d)$. Then, $\mathbf{V}$ is invertible.*

Proof. Take any set of distinct values $x_1, \ldots, x_d \in \mathbb{F}$ and let $\mathbf{V} \in \mathbb{F}^{d \times d}$ be the Vandermonde matrix associated with $(x_1, \ldots, x_d)$. We argue that the kernel of $\mathbf{V}$ is trivial. Suppose $\mathbf{V}\mathbf{f} = \mathbf{0}$ for some $\mathbf{f}^\intercal = [f_0 \mid \cdots \mid f_{d-1}] \in \mathbb{F}^d$. Let $f = \sum_{i \in [d]} f_{i-1} x^{i-1}$ be the polynomial whose coefficients are given by $\mathbf{f}$. By Eq. (5.1), we have that

$$\mathbf{V}\mathbf{f} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_d) \end{bmatrix}.$$

By definition, the polynomial f has degree at most $d-1$, but $f(x_i) = 0$ for all $i \in [d]$. Since the x_i's are distinct, this means the polynomial f has at least d roots. By Fact 5.1, a non-zero polynomial of degree $d-1$ can have at most $d-1$ roots, so this means that the polynomial f is the identically-zero polynomial. In this case, $\mathbf{f} = 0$, and the claim holds (i.e., $\mathbf{V}\mathbf{f} = \mathbf{0}$ if and only if $\mathbf{f} = \mathbf{0}$). Since the kernel of $\mathbf{V}$ is empty, it follows that $\mathbf{V}$ is full rank, and thus, invertible. □

Lagrange interpolation. We now return to the problem of polynomial interpolation. Consider again a sequence of points $(x_1, y_1), \ldots, (x_d, y_d)$ where all the x_i's are distinct. Let $\mathbf{V} \in \mathbb{F}^{d \times d}$ be the Vandermonde matrix associated with $x_1, \ldots, x_d$ and let $\mathbf{y} = [y_1 \mid \cdots \mid y_d]^\intercal$ be the vector of targets. Then, the problem of polynomial interpolation equates to finding a solution to the linear system $\mathbf{V}\mathbf{f} = \mathbf{y}$. By Lemma 5.2, $\mathbf{V}$ is invertible, so we can write $\mathbf{f} = \mathbf{V}^{-1}\mathbf{y}$. If we parse $\mathbf{f} = [f_0 \mid \cdots \mid f_{d-1}]^\intercal$, then the polynomial $f(x) \coloneqq \sum_{i \in [d]} f_{i-1} x^{i-1}$ satisfies $f(x_i) = y_i$ for all $i \in [d]$. The columns of $\mathbf{V}^{-1}$ are often referred to as the coefficients of the *Lagrange interpolation polynomials*. Namely, suppose we write

$$\mathbf{V}^{-1} = \begin{bmatrix} \ell_{1,0} & \ell_{2,0} & \cdots & \ell_{d,0} \\ \ell_{1,1} & \ell_{2,1} & \cdots & \ell_{d,1} \\ \vdots & \vdots & & \vdots \\ \ell_{1,d-1} & \ell_{2,d-1} & \cdots & \ell_{d,d-1} \end{bmatrix} \in \mathbb{F}^{d \times d}.$$

For $i \in [d]$, let $L_i(x) \coloneqq \sum_{j \in [d]} \ell_{i,j-1} x^{j-1}$. From Eq. (5.1), we can write

$$\mathbf{V}\mathbf{V}^{-1} = \begin{bmatrix} L_1(x_1) & L_2(x_1) & \cdots & L_d(x_1) \\ L_1(x_2) & L_2(x_2) & \cdots & L_d(x_2) \\ \vdots & \vdots & & \vdots \\ L_1(x_d) & L_2(x_d) & \cdots & L_d(x_d) \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}.$$

In particular, this means that $L_i(x_j) = 1$ if $i = j$ and $L_i(x_j) = 0$ if $i \neq j$. Then, another way we can express the polynomial $f : \mathbb{F} \to \mathbb{F}$ that interpolates the points $(x_1, y_1), \ldots, (x_d, y_d)$ is

$$f(x) \coloneqq \sum_{i \in [d]} y_i L_i(x).$$

To summarize, we obtain the following corollary to Lemma 5.2:

Corollary 5.3 (Polynomial Interpolation). *Take any collection of points $(x_1, y_1), \ldots, (x_d, y_d) \in \mathbb{F}^2$ where all the x_i's are distinct. Then, there exists a unique polynomial $f : \mathbb{F} \to \mathbb{F}$ of degree at most $d-1$ such that $f(x_i) = y_i$ for all $i \in [d]$.*

Proof. Let f be a polynomial of degree at most $d-1$ whose coefficients are given by a vector $\mathbf{f} \in \mathbb{F}^d$. Let $\mathbf{V} \in \mathbb{F}^{d \times d}$ be the Vandermonde matrix associated with $(x_1, \ldots, x_d)$ and $\mathbf{y} = [y_1 \mid \cdots \mid y_d]^\intercal \in \mathbb{F}^d$. If $f(x_i) = y_i$ for all $i \in [d]$, then $\mathbf{Vf} = \mathbf{y}$. By Lemma 5.2, $\mathbf{V}$ is invertible, so this linear system has a unique solution: $\mathbf{f} = \mathbf{V}^{-1}\mathbf{y}$. □

Shamir secret sharing. Before showing how to construct a t-out-of-n threshold signature scheme, we start with a more basic problem: how to share a secret s with n users such that any t-subset of the parties are able to reconstruct the secret. Moreover, we say the secret sharing scheme satisfies perfect secrecy if every set with fewer than t shares perfectly hide the secret s. We typically refer to the user that generates the shares as the "dealer."

Construction 5.4 (Shamir Secret Sharing). Let $\mathbb{F}$ be a field and take any $n < |\mathbb{F}|$. Let $1 \leq t \leq n$ be a threshold. The t-out-of-n Shamir secret sharing scheme over $\mathbb{F}$ is then defined as follows:

- **Share generation:** To construct a t-out-of-n secret-sharing of a secret $s \in \mathbb{F}$, the dealer samples $\alpha_1, \ldots, \alpha_{t-1} \xleftarrow{R} \mathbb{F}$ and defines the polynomial $f(x) = s + \alpha_1 x + \alpha_2 x^2 + \ldots + \alpha_{t-1} x^{t-1}$. Let $x_1, \ldots, x_n \in \mathbb{F}$ be a collection of distinct non-zero points in $\mathbb{F}$. For each $i \in [n]$, the dealer defines the i^{th} share of the secret to be $s_i = (x_i, f(x_i))$.
- **Reconstruction:** To reconstruct s using a collection of shares $\{(x_i, y_i)\}_{i \in T}$ where $|T| \geq t$, the users interpolate the polynomial $f : \mathbb{F} \to \mathbb{F}$ where $f(x_i) = y_i$ for all $i \in T$. They then recover the secret s by computing $s = f(0)$.

Correctness of Shamir secret sharing. Consider a set of $\{(x_i, y_i)\}_{i \in T}$ shares where $|T| \geq t$. Since each share (x_i, y_i) is a point on some polynomial f (chosen by the dealer) and f has degree at most $t-1$, it is uniquely determined by any collection of t points (Corollary 5.3). Thus, any set of t shares uniquely determine the dealer's polynomial f; in this case, $s = f(0)$ by construction.

Perfect security of Shamir secret sharing. For security, suppose we have fewer than t shares. Without loss of generality, we will assume that we have exactly $t-1$ shares (if the secret cannot be reconstructed with a set of $t-1$ shares, then it also cannot be constructed by any smaller set). Let $\{(x_i, y_i)\}_{i \in T}$ be a collection of shares where $|T| < t$. We will show that the following two distributions are identical:

- **Real distribution:** Sample $\alpha_1, \ldots, \alpha_{t-1} \xleftarrow{R} \mathbb{F}$, let $f(x) = s + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_{t-1} x^{t-1}$, and output the shares $\{(x_i, f(x_i))\}_{i \in T}$.
- **Ideal distribution**: Sample $y_i \xleftarrow{R} \mathbb{F}$ for all $i \in T$ and output $\{(x_i, y_i)\}_{i \in T}$.

Perfect secrecy follows immediately since the shares output by the ideal distribution are independent of the secret s. It suffices to argue that these two distributions are identical. To argue this, consider the mapping

$$\mathsf{Share}_s : (\alpha_1, \ldots, \alpha_{t-1}) \in \mathbb{F}^{t-1} \mapsto [g(x_{i_1}), \ldots, g(x_{i_{t-1}})] \in \mathbb{F}^{t-1} \tag{5.2}$$

where $g(x) := s + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_{t-1} x^{t-1}$ and $T = \{i_1, \ldots, i_{t-1}\} \subset [n]$. Here, the input is a vector of polynomial coefficients, which correspond to the dealer's randomness in the real distribution, and the

output is a set of $t-1$ shares. We claim that for all $s \in \mathbb{F}$, Share_s function is injective. Suppose $\mathsf{Share}_s(\boldsymbol{\alpha}) = \mathsf{Share}_s(\boldsymbol{\alpha}')$ where $\boldsymbol{\alpha}, \boldsymbol{\alpha}' \in \mathbb{F}^{t-1}$. We write $\boldsymbol{\alpha} = [\alpha_1 \mid \cdots \mid \alpha_{t-1}]^\intercal$ and $\boldsymbol{\alpha}' = [\alpha'_1 \mid \cdots \mid \alpha'_{t-1}]^\intercal$ Let $g(x) = s + \sum_{i\in[t-1]} \alpha_i x^i$ and $g'(x) = s + \sum_{i\in[t-1]} \alpha'_i x^i$. Let $h \equiv g - g'$. Since $g(0) = g'(0)$, it follows that $h(0) = 0$. Moreover, since $\mathsf{Share}_s(\boldsymbol{\alpha}) = \mathsf{Share}_s(\boldsymbol{\alpha}')$, we have that $g(x_{i_j}) = g'(x_{i_j})$ for all $j \in [t-1]$. Thus,

$$h(0) = h(x_{i_1}) = \cdots = h(x_{i_{t-1}}) = 0.$$

This means the polynomial h has at least t roots. Since g, g' are polynomials with degree at most $t-1$, the degree of h is also at most $t-1$. By Fact 5.1, this means that h must be the identically-zero polynomial, and so $g = g'$. Since $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$ are the coefficients of g, g', respectively, this means that $\boldsymbol{\alpha} = \boldsymbol{\alpha}'$. We conclude that the Share_s mapping in Eq. (5.2) is injective, and thus bijective. But if Share_s is a bijection, then the distribution of $\mathsf{Share}_s(\boldsymbol{\alpha})$ for $\boldsymbol{\alpha} \xleftarrow{R} \mathbb{F}^{t-1}$ is also uniform over $\mathbb{F}^{t-1}$. This matches the ideal distribution, and the claim holds.

A threshold BLS signature scheme. Suppose we have t Shamir secret sharing shares $(x_1, y_1) \ldots (x_t, y_t)$. The reconstruction algorithm computes the Vandermonde matrix $\mathbf{V}$, interpolates the polynomial f, and output $f(0)$. Here, $f(0)$ is simply the constant coefficient in f (i.e., its first coordinate). Thus, we can combine the second and third steps into a single equation:

$$s = \mathbf{e}_1^\intercal \mathbf{f} = \mathbf{e}_1^\intercal \mathbf{V}^{-1} \mathbf{y},$$

where $\mathbf{e}_1$ denote the first canonical basis vector $\mathbf{e}_1 = [1 \mid 0 \mid \cdots \mid 0]^\intercal$, and $\mathbf{y}$ denotes the vector of shares.

Observe that this equation is linear in terms of the shares $\mathbf{y}$. This means we can convert the equation to a dot product $s = \boldsymbol{\lambda}^\intercal \mathbf{y}$, where the reconstruction coefficients $\boldsymbol{\lambda}$ is $\boldsymbol{\lambda} \coloneqq \mathbf{e}_1^\intercal \mathbf{V}^{-1}$. Applying this to the BLS signature scheme, we get the following equation:

$$\sigma = H(m)^s = H(m)^{\boldsymbol{\lambda}^\intercal \mathbf{y}} = H(m)^{\sum_{i\in T} \lambda_i y_i} = \prod_{i\in T} H(m)^{\lambda_i y_i}$$

The $H(m)^{y_i}$ computation resembles the standard BLS signing algorithm, using y_i as the signing key for each party; since y_i is the secret share for party i, this is feasible to use as the per-party signing algorithm. λ_i is public and can be recomputed knowing only which parties are signing the message, so that step of the signature computation can be used to combine shares. This gives us the full BLS t-out-of-n threshold signing algorithm:

Definition 5.5 (BLS t-out-of-n threshold signature algorithm)**.** The BLS t-out-of-n threshold signing algorithm is defined as follows:

- Setup: Choose a random BLS signing key $k \leftarrow F_p$. Using the Shamir secret sharing algorithm, split k into N shares $(k_1 ... k_N) \in Z_p$. Give party i share (i, k_i).
- $\mathsf{Sign}((i, k_i), m)$: Let $\sigma_i = m^{k_i}$. Output (i, σ_i).
- $\mathsf{Aggregate}(\{(x_1, \sigma_1), ..., (x_t, \sigma_t)\})$: Compute the reconstuction coefficients λ_i using the x_i values. Output $\sigma = \prod_{i\in[t]} \sigma_i^{\lambda_i}$.

The scheme is correct because $\sigma = \prod_{i\in[t]} \sigma_i^{\lambda_i} = \prod_{i\in[t]} m^{k_i \lambda_i} = m^{\sum_{i\in[t]} k_i \lambda_i} = m^s$. Security follows from the Shamir secret sharing proof and from BLS signature security.

CS 395T: Topics in Cryptography February 1, 2024

Lecture 6: Identity-Based Encryption

Lecturer: David Wu *Scribe: Sidh Suchdev*

In an ordinary public key encryption system, every user has their own public key and secret key. Communication with many other users requires keeping track of all of those users' public keys, which can take up lots of space. Identity-based encryption is designed to reduce the amount of space needed to keep track of many public keys by dynamically generating them from an arbitrary identifier (like an email address) and a single master public key.

Definition 6.1 (Identity-based Encryption Scheme)**.** An identity-based encryption scheme with message space $\mathcal{M}$ is a tuple of efficient algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk})$: On input a security parameter λ, the setup algorithm outputs a master public key mpk (used for encryption) and a master secret key msk (used to issue keys).
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$: On input the master secret key msk and an identity id, the key-generation algorithm outputs an identity key $\mathsf{sk}_{\mathsf{id}}$.
- $\mathsf{Encrypt}(\mathsf{mpk}, \mathsf{id}, m) \to \mathsf{ct}$: On input the master public key mpk, a target identity id, and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext. (In some settings, this can be thought of as two separate algorithms: one that takes mpk and id and outputs a per-user public key, and a second that uses the per-user public key to encrypt a message.)
- $\mathsf{Decrypt}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct}) \to m$: On input an identity key $\mathsf{sk}_{\mathsf{id}}$ and a ciphertext ct, the decryption algorithm either outputs a message $m \in \mathcal{M}$.

Correctness. The scheme is correct if for all identities id and all messages $m \in \mathcal{M}$, it holds that

$$\Pr\left[\mathsf{Decrypt}(\mathsf{mpk}, \mathsf{id}, \mathsf{ct}) = m : \begin{array}{c}(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, \mathsf{id}, m)\end{array}\right] = 1.$$

Security. We now define semantic security for an IBE scheme. Intuitively, security says that the adversary cannot learn anything about a message encrypted to an identity id^* if it does not possess a secret key for id^*. We now define this formally. As usual, the game is parameterized by a bit $b \in \{0, 1\}$.

- **Setup:** At the beginning of the game, the challenger samples $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ and sends mpk to the adversary $\mathcal{A}$.
- **Key-generation queries:** The adversary $\mathcal{A}$ is now allowed to issue key-generation queries on identities id of its choosing. On each query, the challenger responds with the secret key $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$.
- **Challenge query:** When the adversary is done making key-generation queries, it outputs a challenge identity id^* and two messages m_0^*, m_1^*. The challenger responds with $\mathsf{ct}_b^* \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, \mathsf{id}^*, m_b^*)$.

- **Key-generation queries:** The challenger can continue to make key-generation queries on identities of its choosing. The challenger responds to each query exactly as before.
- **Output:** At the end of the experiment, the adversary outputs a bit $b' \in \{0,1\}$.

We say the adversary $\mathcal{A}$ is admissible for the semantic security game if it does not make a key-generation query on the challenge identity id^*. Then, we say the IBE scheme is semantically secure if for every efficient adversary $\mathcal{A}$,

$$|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| \leq \mathsf{negl}(\lambda).$$

(This scheme can also be extended to IND-CCA security by adding decryption queries for an arbitrary identity and ciphertext. In the IND-CCA version, an attacker is admissible as long as it does not make any decryption queries for ct^* on id^*.

Boneh-Franklin IBE. We now introduce the IBE scheme by Boneh and Franklin [BF01]. The structure of this scheme shares many similarities with the BLS signature scheme (Construction 3.2). In particular, the secret key for an identity id is a BLS signature on the identity. In this sense, the Boneh-Franklin IBE can also be viewed as a "witness encryption" scheme [GGSW13] for the BLS signature language; decrypting a ciphertext for an identity id requires knowledge of a BLS signature for id.[1]

Construction 6.2 (Boneh-Franklin Identity Based Encryption)**.** Let $(\mathbb{G}, \mathbb{G}_T, e)$ be a (symmetric) pairing group and let g be a generator of $\mathbb{G}$. Let $\mathcal{H}\colon \{0,1\}^n \to \mathbb{G}$ be a hash function. We define the identity based encryption scheme below:

- $\mathsf{Setup}(1^\lambda)$: Sample $k \xleftarrow{R} \mathbb{Z}_p$ and output $\mathsf{mpk} = g^k, \mathsf{msk} = k$
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$: On input the master secret key $\mathsf{msk} = k$, outputs $\mathsf{sk}_{\mathsf{id}} = \mathcal{H}(\mathsf{id})^k$
- $\mathsf{Encrypt}(\mathsf{mpk}, \mathsf{id}, m)$: Sample $r \xleftarrow{R} \mathbb{Z}_p$ and output $\mathsf{ct} = (g^r, e(\mathsf{mpk}, \mathcal{H}(\mathsf{id}))^r \cdot m)$
- $\mathsf{Decrypt}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct})$: On input the secret key $\mathsf{sk}_{\mathsf{id}}$ and the ciphertext $\mathsf{ct} = (u, v)$, output $v/e(u, \mathsf{sk}_{\mathsf{id}})$

Correctness. We can verify the correctness by a simple calculation. Take any identity id and message m. Let $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$. Let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, \mathsf{id}, m)$ and $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$. Write $\mathsf{mpk} = g^k$ and $\mathsf{msk} = k$. Then, $\mathsf{sk}_{\mathsf{id}} = \mathcal{H}(\mathsf{id})^k$. Next, write $\mathsf{ct} = (u, v)$ where $u = g^r$ and $v = e(g^k, \mathcal{H}(\mathsf{id}))^r \cdot m$. Then, using bilinearity, we have

$$\mathsf{Decrypt}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct}) = \frac{v}{e(u, \mathsf{sk}_{\mathsf{id}})} = \frac{e(g^k, \mathcal{H}(\mathsf{id})^r) \cdot m}{e(g^r, \mathcal{H}(\mathsf{id})^k)} = \frac{e(g, \mathcal{H}(\mathsf{id}))^{kr}}{e(g, \mathcal{H}(\mathsf{id}))^{kr}} \cdot m = m.$$

Remark 6.3 (Parallels with ElGamal)**.** The structure of the Boneh-Franklin IBE scheme shares many similarities with that of ElGamal encryption. Recall first that in ElGamal encryption, the public key is (g, h) and the secret key is the discrete log $k \in \mathbb{Z}_p$ where $h = g^k$. An encryption of $m \in \mathbb{G}$ is the pair $\mathsf{ct} = (g^r, h^r \cdot m)$. To decrypt, the user relies on the fact that $(g^r)^k = h^r$, which suffices to recover m. Knowledge of the key $k \in \mathbb{Z}_p$ is used to "multiply-by-k in the exponent" (to translate from g^r to $g^{kr} = h^r$).

In Boneh-Franklin, we can interpret the quantity $e(\mathsf{mpk}, \mathcal{H}(\mathsf{id})) = e(g^k, \mathcal{H}(\mathsf{id}))$ as the public key $\mathsf{pk}_{\mathsf{id}}$ associated with the identity id. Observe now that the encryption algorithm in Boneh-Franklin is

[1] Note that this high-level description does not map to a formal definition of witness encryption, but it does share conceptual similarities.

basically constructing an ElGamal ciphertext with respect to $\mathsf{pk}_{\mathsf{id}}$: that is, sample $r \xleftarrow{R} \mathbb{Z}_p$ and outputs $\mathsf{ct} = (g^r, \mathsf{pk}_{\mathsf{id}}^r \cdot m)$. To decrypt, the *pairing* is used to implement the "multiply-by-k in the exponent" operation needed for the ElGamal-style decryption procedure:

$$e(g^r, \mathsf{sk}_{\mathsf{id}}) = e(g^r, \mathcal{H}(\mathsf{id})^k) = e(g^k, \mathcal{H}(\mathsf{id}))^r = \mathsf{pk}_{\mathsf{id}}^r.$$

Security. To prove security, we rely on the DBDH problem previously defined in Lecture 2, where we are to distinguish $(g, g^x, g^y, g^z, g^{xyz}),\ x, y, z \xleftarrow{R} \mathbb{Z}_p$ from $(g, g^x, g^y, g^z, g^r),\ x, y, z, r \xleftarrow{R} \mathbb{Z}_p$.

We will prove a stronger statement than semantic security. We will prove that the ciphertext generated by the Boneh-Franklin scheme is indistinguishable from random under the random oracle model. Let D_0 be the distribution of ciphertexts generated by the challenger and let D_1 be the distribution of random ciphertexts. An adversary wins if it can successfully distinguish between the two distributions.

We will do this by a reduction to the DBDH problem. Let $\mathcal{A}$ be an adversary for the game described before. Without loss of generality, we assume the following about $\mathcal{A}$:

1. $\mathcal{A}$ makes at most Q queries to the random oracle, where $Q = \mathsf{poly}(\lambda)$
2. $\mathcal{A}$ always queries the random oracle on id before asking for the key on id
3. $\mathcal{A}$ always queries the random oracle on id^* before outputting the challenge
4. $\mathcal{A}$ always queries the random oracle any id at most once

Note, similarly to BLS signatures, that these assumptions follow naturally and that if any adversary does not follow these, we can construct an adversary $\mathcal{A}'$ where these assumptions hold.

Now we construct adversary $\mathcal{B}$ as follows:

- On input $(g, u = g^x, v = g^y, w = g^z, T)$, algorithm $\mathcal{B}$ sets $\mathsf{mpk} = u$ and $i^* \xleftarrow{R} [Q]$, where i^* is the guess of the challenge query index.
- We run algorithm $\mathcal{A}$ with our mpk, and respond to the following queries
  - $\mathcal{A}$ makes a random oracle query on id:
    - If $i = i^*$, then output $\mathcal{H}(\mathsf{id}) = \mathsf{sk}_{\mathsf{id}} = v$
    - Otherwise, sample $\alpha_{\mathsf{id}} \xleftarrow{R} \mathbb{Z}_p$ and output $g^{\alpha_{\mathsf{id}}}$.
  - $\mathcal{A}$ makes a key generation query on id:
    - If $i = i^*$, abort
    - Otherwise, output $u^{\alpha_{\mathsf{id}}}$
  - $\mathcal{A}$ makes a challenge query m and id^*:
    - If $i = i^*$, then output $\mathsf{ct}^* = (w, T)$
    - Otherwise, abort
- $\mathcal{A}$ outputs bit $b' \in \{0, 1\}$, which $\mathcal{B}$ forwards to the challenger.

Suppose our guess of i^* is correct and $T = e(g, g)^{xyz}$. Then the following holds:

$$\begin{aligned} T \cdot m &= e(g, g)^{xyz} \cdot m \\ &= e(g^x, g^y)^z \cdot m \\ &= e(\mathsf{mpk}, \mathcal{H}(\mathsf{id}))^z \cdot m \end{aligned}$$

We now analyze the advantage of our adversary $\mathcal{B}$. Suppose $\mathsf{Adv}_{\mathcal{A}} = \epsilon$. Note that when i^* is guessed incorrectly, adversary $\mathcal{B}$ aborts, but when when i^* is guessed correctly then the advantage of $\mathcal{B}$ is at least ϵ. Thus, $\mathsf{Adv}_{\mathcal{B}} \geq \epsilon/Q >$ negl.

Remark 6.4 (Challenges in Proving Adaptive Security)**.** The strategy to sidestep the paradox of an unknown sk_{id} to prove adaptive security is known as partitioning. The other prominent method of showing adaptive security is the dual-system, developed by Waters.

Limitations of IBE. Identity based encryption provides a neat solution to the issue of compressing multiple public and secret keys for users. However, an important consideration when using identity based encryption is that it changes the threat model in public key encryption. For example, suppose we have users Alice and Bob. In standard public key cryptography, Alice generates $(\mathsf{pk}_A, \mathsf{sk}_B)$, Bob generates $(\mathsf{pk}_B, \mathsf{sk}_B)$ and they publish $\mathsf{pk}_A, \mathsf{pk}_B$. Note that only Alice knows her secret key sk_A and only Bob knows his secret key sk_B. In identity based encryption, a third party authority generates the $\mathsf{mpk}, \mathsf{msk}$, sends Alice her sk_A and Bob his sk_B, and publishes the mpk. In this case, the third party knows *every* user's secret key. This might be undesirable since the third party authority is a single point of failure, where if it were ever compromised, all communications between parties could be decrypted. We can avoid this situation through a scheme called registration-based encryption (RBE), where the authority instead aggregates identities and public keys, with no knowledge of the secret keys associated with them.

CS 395T: Topics in Cryptography February 6, 2024

Lecture 7: Broadcast Encryption

Lecturer: David Wu *Scribe: Matthew Healy*

In the last lecture, we introduced identity-based encryption, a generalization of public-key encryption that allows users to encrypt messages to an arbitrary identity. An IBE scheme can be viewed as "compressing" an *exponential* number of public keys for different identities into a single set of public parameters. In this lecture, we introduce the concept of broadcast encryption, which allows a broadcaster to encrypt a message to a set of receivers with a ciphertext whose size scales sublinearly with the number of recipients.

Definition 7.1 (Broadcast Encryption)**.** A broadcast encryption scheme with message space $\mathcal{M}$ is a tuple of efficient algorithms $(\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input the security parameter λ and the number of users N, the setup algorithm outputs a master public key mpk and a collection of N decryption keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$.
- $\mathsf{Encrypt}(\mathsf{mpk}, S, m)$: On input the master public key mpk, a "broadcast set" $S \subseteq [N]$, and a message m, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}_i, S, \mathsf{ct})$: On input a decryption key sk_i, the broadcast set $S \subseteq [N]$, and a ciphertext ct, the decryption algorithm outputs a message m.

Correctness. The scheme is correct if for all integers $N \in \mathbb{N}$, all sets $S \subseteq [N]$, and all indices $i \in S$, it holds that

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}_i, S, \mathsf{ct}) = m : \begin{array}{c}(\mathsf{mpk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N) \leftarrow \mathsf{Setup}(1^\lambda, N) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, S, m)\end{array}\right] = 1.$$

Succinctness. A trivial broadcast encryption scheme is to concatenate N ciphertexts (under N independent public keys) together. The goal in broadcast encryption is to achieve *short* ciphertext. Specifically, we say the broadcast encryption scheme satisfies succinctness if $|\mathsf{ct}| = o(N) \cdot \mathsf{poly}(\lambda)$ where N is the number of users and λ is the security parameter.

Security. We now define semantic security for a broadcast encryption scheme. Intuitively, security says that the adversary cannot learn anything about a message encrypted to a set $S \subseteq [N]$ if it does not possess a secret key for some index $i \in S$. We now define this formally. As usual, the game is parameterized by a bit $b \in \{0, 1\}$.

- **Setup:** At the beginning of the game, the adversary declares the number of users 1^N. The challenger then samples $(\mathsf{mpk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N) \leftarrow \mathsf{Setup}(1^\lambda, N)$ and sends mpk to the adversary $\mathcal{A}$.
- **Key-generation queries:** The adversary $\mathcal{A}$ can now ask for secret keys on indices $i \in [N]$ of its choosing. On each query $i \in [N]$, the challenger responds with sk_i.
- **Challenge query:** When the adversary is done making key-generation queries, it outputs a set $S^* \subseteq [N]$ and two messages m_0^*, m_1^*. The challenger responds with $\mathsf{ct}_b^* \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, S^*, m_b^*)$.

- **Output:** At the end of the experiment, the adversary outputs a bit $b' \in \{0,1\}$.

We say the adversary $\mathcal{A}$ is admissible for the semantic security game if it does not make a key-generation query on any index $i \in S^*$ Then, we say the broadcast encryption scheme is semantically secure if for every efficient adversary $\mathcal{A}$,

$$|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| \leq \mathsf{negl}(\lambda).$$

Boneh-Gentry-Waters broadcast encryption scheme. We now describe the broadcast encryption scheme by Boneh, Gentry, and Waters [BGW05] The key idea underlying this scheme is to use *powers in the exponent*. The "correlation" structure induced by the powers will play a role both in correctness and for arguing security. We build this scheme up step-by-step before giving the formal description in Construction 7.2.

- **Public parameters:** The public parameters in BGW will contain group elements of the form

$$g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^N}, g^{\alpha^{N+2}}, \ldots, g^{\alpha^{2N}},$$

  where $\alpha \xleftarrow{R} \mathbb{Z}_p$ is a random exponent. The first set of N powers $\left(g^{\alpha}, \ldots, g^{\alpha^N}\right)$ function as the "public keys" associated with users $1, \ldots, N$ respectively, while the latter set of elements $\left(g^{\alpha^{N+2}}, \ldots, g^{\alpha^{2N}}\right)$ are "helper components" that are used for decryption. Critically, there is a *hole* at $g^{\alpha^{N+1}}$. The missing component will be used to "encrypt."

- **Encryption:** To encrypt a message $m \in \mathbb{G}_T$, sample $r \xleftarrow{R} \mathbb{Z}_p$ and compute

$$\mathsf{ct} = \left(g^r, e(g^{\alpha}, g^{\alpha^N})^r \cdot m\right).$$

  Here, $e(g^{\alpha}, g^{\alpha^N})^r$ is the key-encapsulation mechanism (KEM) and can be rewritten as $e(g^{\alpha}, g^{\alpha^N}) = e(g, g^{\alpha^{N+1}})$. Note that $g^{\alpha^{N+1}}$ cannot be given out, as otherwise anyone can compute $e(g^r, g^{\alpha^{N+1}})$.

- **Decryption.** However, with just the encryption construction stated above, we are unable to decrypt the ciphertext. We will need to include additional information in the ciphertext to allow a party to compute $e(g^{\alpha}, g^{\alpha^N})^r$ in order to support decryption. We also want to ensure that a party i should only be able to decrypt if $i \in S$; if $i \notin S$, it should not be possible to compute $e(g, g^{\alpha^{N+1}})^r$. The approach to generating such a ciphertext is as follows:
  - Let $g_i = g^{\alpha^i}$. Observe that $e(g_i, g_j) = e(g^{\alpha^i}, g^{\alpha^j}) = e(g,g)^{\alpha^i \alpha^j} = e(g,g)^{\alpha^{i+j}} = e(g, g_{i+j})$.
  - In the ciphertext, include $\prod_{j \in S} g_{N+1-j}^r$. Now, take any index $i \in S$. By bilinearity, we now have

$$\begin{aligned} e\left(g_i, \prod_{j \in S} g_{N+1-j}^r\right) &= \prod_{j \in S} e(g_i, g_{N+1-j})^r \\ &= e(g_i, g_{N+1-i})^r \cdot \prod_{j \in S \setminus \{i\}} e(g_i, g_{N+1-j})^r \\ &= e(g, g_{N+1})^r \cdot \prod_{j \in S \setminus \{i\}} e(g, g_{N+1-j+i})^r \end{aligned} \tag{7.1}$$

- Note that $e(g, g_{N+1})^r$ is our quantity of interest and

$$e(g, g_{N+1-j+i})^r = e(g^r, g_{N+1-j+i}).$$

  When $i, j \in [N]$, then $2 \leq N+1-j+i \leq 2N$. Moreover, if $i \neq j$, then $N+1-j+i \neq N+1$. This means $g_{N+1-j+i}$ is contained within the public parameters.
- Conversely, if $i \notin S$, then $e(g_i, \prod_{j \in S} g_{N+1-j}^r)$ does not contain the critical term $e(g, g_{N+1})^r$.

- **Decryption keys.** At this point, we have shown that knowledge of g_i is sufficient to decrypt (whenever $i \in S$). However, g_i is public so *anyone* could decrypt at this point. To address this, we introduce introduce an additioanl blinding factor to the ciphertexts that can only be removed by a user who possesses a secret key. Specifically, the setup algorithm now samples $\gamma \xleftarrow{R} \mathbb{Z}_p$ and sets the secret key for user i to be $d_i := g_i^\gamma$. The public key will then contain $v = g^\gamma$. Now, in the ciphertext, we replace the element $\prod_{j \in S} g_{N+1-j}^r$ with the element

$$\left[v \cdot \prod_{j \in S} g_{N+1-j} \right]^r.$$

  Consider the updated version of the decryption relation from Eq. (7.1):

$$\begin{aligned} e\left(g_i, v^r \cdot \prod_{j \in S} g_{N+1-j}^r \right) &= e(g_i, v^r) \cdot \prod_{j \in S} e(g_i, g_{N+1-j})^r \\ &= \boxed{e(g_i, v^r)} \cdot e(g, g_{N+1})^r \cdot \prod_{j \in S \setminus \{i\}} e(g, g_{N+1-j+i})^r. \end{aligned}$$

  Notably, the decryption relation from Eq. (7.1) now has an *additional* term $e(g_i, v^r)$. Cancelling out this term will require knowledge of the i^{th} user's decryption key. Namely, observe that

$$e(g_i, v^r) = e(g_i, g^{\gamma r}) = e(g_i^\gamma, g^r) = e(d_i, g^r).$$

  Since g^r is part of the ciphertext, user i can compute $e(d_i, g^r)$ and cancel out this term.

Putting all the pieces together, we now arrive at the full Boneh-Gentry-Waters construction:

Construction 7.2 (Boneh-Gentry-Waters Broadcast Encryption [BGW05])**.** Using the approach described above, the broadcast encryption scheme is implemented as follows:

- $\mathsf{Setup}(1^\lambda, 1^N)$: Sample $\alpha \xleftarrow{R} \mathbb{Z}_p$ and $\gamma \xleftarrow{R} \mathbb{Z}_p$. Let $g_i = g^{\alpha^i}$ and $v = g^\gamma$. Output the master public key

$$\mathsf{mpk} = (g, g_1, g_2, \ldots, g_N, g_{N+2}, \ldots, g_{2N}, v),$$

  and the decryption keys $\mathsf{sk}_i = (i, g_i, d_i)$ where $d_i = g_i^\gamma$.

- $\mathsf{Encrypt}(\mathsf{mpk}, S, m)$: On input the master public key $\mathsf{mpk} = (g, g_1, g_2, \ldots, g_N, g_{N+2}, \ldots, g_{2N}, v)$, a set $S \subseteq [N]$, and a message $m \in \mathbb{G}_T$, sample $r \xleftarrow{R} \mathbb{Z}_p$ and output the ciphertext

$$\mathsf{ct} = \left(g^r \ , \ v^r \cdot \prod_{j \in S} g_{N+1-j}^r \ , \ e(g_1, g_N)^r \cdot m \right).$$

- $\mathsf{Decrypt}(\mathsf{sk}_i, S, \mathsf{ct})$: On input a decryption key $\mathsf{sk}_i = (i, g_i, d_i)$, the set $S \subseteq [N]$, and a ciphertext $\mathsf{ct} = (C_1, C_2, C_3)$, output

$$z = C_3 \cdot \frac{e\big(d_i \cdot \prod_{j \in S\setminus\{i\}, C_1} g_{N+1-j+i}\big)}{e(g_i, C_2)}.$$

Correctness. We now argue correctness. Let $\mathsf{mpk} = (g, g_1, \ldots, g_N, g_{N+2}, \ldots, g_{2N}, v)$, $\mathsf{sk} = (i, g_i, d_i)$ be the master public key and secret keys output by $\mathsf{Setup}(1^\lambda, N)$. Let $\mathsf{ct} = (C_1, C_2, C_3)$ be an encryption of a message m to the set S under mpk. By construction, $g_i = g^{\alpha^i}$, $d_i = g_i^\gamma$, and $v = g^\gamma$. Consider now the decryption relation:

$$\begin{aligned}
e(g_i, C_2) &= e\left(g_i, v \cdot \prod_{j \in S} g_{N+1-j}\right)^r \\
&= e(g_i, g_{N+1-i})^r \cdot e(g_i, v)^r \cdot e\left(g_i, \prod_{j \in S\setminus\{i\}} g_{N+1-j}\right)^r \\
&= e(g, g_{N+1})^r \cdot e(g_i, v)^r \cdot \prod_{j \in S\setminus\{i\}} e(g_i, g_{N+1-j})^r \\
e\left(d_i \prod_{j \in S\setminus\{i\}} g_{N+1-j+i}, C_1\right) &= e(d_i, g^r) \cdot \prod_{j \in S\setminus\{i\}} e(g_{N+1-j+i}, g^r) \\
&= e(g_i^\gamma, g)^r \cdot \prod_{j \in S\setminus\{i\}} e(g_i, g_{N+1-j})^r \\
&= e(g_i, v)^r \cdot \prod_{j \in S\setminus\{i\}} e(g_i, g_{N+1-j})^r
\end{aligned}$$

Thus,

$$\frac{e(g_i, C_2)}{e\big(d_i \prod_{j \in S\setminus\{i\}} g_{N+1-j+i}, C_1\big)} = e(g, g_{N+1})^r.$$

Correspondingly,

$$C_3 \cdot \frac{e\big(d_i \prod_{j \in S\setminus\{i\}} g_{N+1-j+i}, C_1\big)}{e(g_i, C_2)} = \frac{m \cdot e(g, g_{N+1})^r}{e(g, g_{N+1})^r} = m.$$

Security. The security of this construction will rely on the N-bilinear Diffie-Hellman exponent (N-BDHE) assumption. We will introduce this assumption and give its proof of security in the following lecture.

CS 395T: Topics in Cryptography February 8, 2024

Lecture 8: Distributed Broadcast Encryption

Lecturer: David Wu *Scribe: Alex Huang*

In the previous lecture, we introduced the Boneh-Gentry-Waters broadcast encryption scheme [BGW05]. In this lecture, we will prove the security of the scheme and show how to adapt it to obtain a distributed broadcast encryption scheme where each recipient can choose their *own* key (instead of relying on a central key distributor).

The N-bilinear Diffie-Hellman Exponent assumption. The security of the BGW broadcast encryption scheme is based on the N-bilinear Diffie-Hellman exponent (N-BDHE) assumption which states that the following two distributions are computationally indistinguishable:

- Sample $\alpha \xleftarrow{R} \mathbb{Z}_p$ and $\beta \xleftarrow{R} \mathbb{Z}_p$. Output

$$\left(g, g^{\beta}, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^N}, g^{\alpha^{N+2}}, \ldots, g^{\alpha^{2N}}, \boxed{e(g,g)^{\alpha^{N+1}\beta}} \right).$$

- Sample $\alpha \xleftarrow{R} \mathbb{Z}_p$, $\beta \xleftarrow{R} \mathbb{Z}_p$ and $r \xleftarrow{R} \mathbb{Z}_p$. Output

$$\left(g, g^{\beta}, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^N}, g^{\alpha^{N+2}}, \ldots, g^{\alpha^{2N}}, \boxed{e(g,g)^{r}} \right).$$

This assumption is similar to the DBDH problem defined in Lecture 2 except that the exponents are α^{N+1} and β and the adversary is given additional powers $g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^N}, g^{\alpha^{N+2}}, \ldots, g^{\alpha^{2N}}$. It is often called a "$q$-type assumption" since the size of the assumption depends on a scheme parameter, in this case the number of users N.

Remark 8.1 (Hardness of N-BDHE)**.** Note that if $p-1$ (or $p+1$) has a factor $t \leq N$, then there is an algorithm for computing α from $(g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^N})$ in time $\tilde{O}(\sqrt{p/t} + \sqrt{t})$ [BG04, Che06]. The generic discrete log algorithm runs in time $\tilde{O}(\sqrt{p})$, so the hardness of the N-BDHE assumption degrades with the size of the assumption.

Selective security. We will prove selective security of the BGW broadcast encryption scheme (Construction 7.2). In the selective security game, the adversary has to declare its challenge set $S \subseteq [N]$ at the *beginning* of the security game before it sees the public parameters. In the usual security game, the adversary can *adaptively* choose the challenge set S after it sees the public parameters and the secret keys (on users of its choosing). Selective security is a much weaker (and less natural) security notion, but enables us to prove security. Intuitively, the scheme is selectively secure as long as the broadcast sets are chosen "independently" of the public parameters and individual users' decryption keys. We do not know how to prove adaptive security of the BGW scheme. At the same time, we also do not know of any attacks on the adaptive security of the BGW scheme.

Definition 8.2 (Selective Security for Broadcast Encryption)**.** For a bit $b \in \{0,1\}$, we define the selective security game for broadcast encryption between a challenger and an adversary $\mathcal{A}$ as follows:

1. The adversary starts by declaring its challenge set $S \subseteq [N]$ and sends it to the challenger.
2. The challenger runs $(\mathsf{mpk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$. The challenger gives mpk and $\{\mathsf{sk}_i\}_{i \notin S}$ to the adversary.
3. The adversary outputs two messages m_0 and m_1.
4. The challenger sends the encryption $\mathsf{ct}_b \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, S, m_b)$ to the adversary.
5. The adversary outputs a bit $b' \in \{0, 1\}$.

The scheme is said to be selectively secure if for all efficient adversaries $\mathcal{A}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

Sketch of security proof. We now give a sketch of the proof of selective security for Construction 7.2 to the N-BDHE assumption. In our sketch, we will show that the challenge ciphertext ct_b in the selective security game (Definition 8.2)

$$\mathsf{ct}_b = \left(g^r \ , \ v^r \cdot \prod_{j \in S} g_{N+1-j}^r \ , \ \boxed{e(g_1, g_N)^r \cdot m_b} \right). \tag{8.1}$$

is computationally indistinguishable from

$$\mathsf{ct}' = \left(g^r \ , \ v^r \cdot \prod_{j \in S} g_{N+1-j}^r \ , \ \boxed{e(g_1, g_N)^t} \right), \tag{8.2}$$

where $t \xleftarrow{R} \mathbb{Z}_p$. In particular, ct' is *independent* of m_0 and m_1. This property implies selective security by a hybrid argument. At the beginning of the game, the reduction algorithm is given the N-BDHE challenge:

$$(h, g, g_1, \ldots, g_N, g_{N+2}, \ldots, g_{2N}, T),$$

where $h = g^\beta$, $g_i = g^{\alpha^i}$, and T is either $e(g, g)^{\alpha^{N+1}\beta}$ or $e(g, g)^t$, and $\alpha, \beta, t \xleftarrow{R} \mathbb{Z}_p$. At the beginning of the game, the reduction algorithm runs the adversary $\mathcal{A}$ to receive the challenge set $S \subseteq [N]$. The reduction algorithm now needs to simulate the master public key mpk, the secret keys sk_i for all $i \notin S$, and the challenge ciphertext ct_b. Rather than simply give the reduction, we provide a step-by-step outline of how the different pieces come together.

- Write the challenge ciphertext as $\mathsf{ct}^* = (\mathsf{ct}_1^*, \mathsf{ct}_2^*, \mathsf{ct}_3^*)$. The main difference between the ciphertext distribution in Eq. (8.1) and Eq. (8.2) is the distribution of ct_3^*. A natural choice then is to use the challenge term T to simulate ct_3^*. For instance, the reduction algorithm could set $\mathsf{ct}_3^* = m_b \cdot T$. In this case, we would like to map $T = e(g, g)^{\alpha^{N+1}\beta}$ to the distribution in Eq. (8.1) and $T = e(g, g)^t$ to the distribution in Eq. (8.2).
- If $m_b \cdot e(g, g)^{\alpha^{N+1}\beta}$ is supposed to coincide with the distribution in Eq. (8.1), then we also need $\mathsf{ct}_1^* = g^\beta$ and $\mathsf{ct}_2^* = v^\beta \cdot \prod_{j \in S} g_{N+1-j}^\beta$. Similarly, the components g_i of the master public key mpk must satisfy $g_i = g^{\alpha^i}$. Thus, we use the challenge terms $g_1, \ldots, g_N, g_{N+2}, \ldots, g_{2N}$ to simulate the components of mpk and set $\mathsf{ct}_1^* = h = g^\beta$.

- The question then is how to construct ct_2^*. The problem is the reduction algorithm does *not* know α or β (i.e., these are the exponents from the challenge). Luckily, we have one remaining degree of freedom! The reduction algorithm has not defined the value of v in the master public key yet. In the real scheme (Construction 7.2), the element v was a random element in $\mathbb{G}$, so we have some flexibility in choosing v. The reduction will choose v so that it is able to simulate $\mathsf{ct}_2^* = v^\beta \cdot \prod_{j \in S} g_{N+1-j}^\beta$ without knowledge of β. One way to do this is to sample $\rho \xleftarrow{R} \mathbb{Z}_p$ and set

$$v = \frac{g^\rho}{\prod_{j \in S} g_{N+1-j}}. \tag{8.3}$$

  For this choice of v, the value of ct_2^* becomes

$$\mathsf{ct}_2^* = v^\beta \cdot \prod_{j \in S} g_{N+1-j}^\beta = \left(v \cdot \prod_{j \in S} g_{N+1-j} \right)^\beta = g^{\rho\beta} = (g^\beta)^\rho = h^\rho.$$

  Observe that the reduction can now simulate ct_2^* *without* knowledge of β. Moreover, v is still a uniformly random group element (since the reduction algorithm sampled $\rho \xleftarrow{R} \mathbb{Z}_p$). At this point, the reduction algorithm has fixed all of the components of the master public key (and the elements of the challenge ciphertext)

- It remains to show that the reduction algorithm can simulate the decryption keys for all indices $i \in [N] \setminus S$. From Construction 7.2, the decryption key for an index i has the form $\mathsf{sk}_i = (i, g_i, d_i)$, where $d_i = g_i^\gamma$. Here γ is the discrete log of v base g (i.e., $v = g^\gamma$). This again seems problematic as the reduction algorithm constructed v *without* knowledge of its discrete log. Here, we make the following observation:

$$d_i = g_i^\gamma = g^{\alpha^i \gamma} = (g^\gamma)^{\alpha^i} = v^{\alpha^i}.$$

  A priori, it may not seem that we have made much progress since the reduction algorithm also does not know α. However, using the definition of v from Eq. (8.3), we have

$$d_i = v^{\alpha^i} = \frac{g^{\rho\alpha^i}}{\prod_{j \in S} g_{N+1-j}^{\alpha^i}} = \frac{g_i^\rho}{\prod_{j \in S} g_{N+1-j+i}}.$$

  Since the reduction algorithm knows the exponent ρ as well as g_i for all $i \neq N+1$, the reduction algorithm can compute d_i as long as $N+1-j+i \neq N+1$ for all $j \in S$. This is true as long as $i \neq j$ for all $j \in S$, or equivalently, if $i \notin S$. This is *precisely* the set of decryption keys the reduction needs to generate. As such, the reduction algorithm is now able to generate decryption keys for every index $i \notin S$.

Before we present the formal reduction, it is instructive to take a step back and study the above reduction strategy. First, we should affirm that the reduction algorithm is *unable* to generate decryption keys for indices $i \in S$. Otherwise, the reduction algorithm could use the key sk_i to decrypt the challenge ciphertext itself and trivially determine whether the ciphertext was distributed according to the distribution in Eq. (8.1) or in Eq. (8.2). As we can see from the above analysis, the requirement $i \notin S$ was critical for being able to generate sk_i (specifically, generating the d_i component of sk_i). The reduction algorithm in this case has essentially *programmed* the challenge set S into the public parameters (without the adversary being able to notice) in a way that allows it to generate decryption keys for all indices $i \notin S$. This type of programming is essential for completing the security proof (and also the reason why the current strategy only suffices for proving selective security). We now state the formal theorem and security reduction.

Theorem 8.3 (Selective Security). *If the N-BDHE assumption holds with respect to $(\mathbb{G}, \mathbb{G}_T, e)$, then Construction 7.2 is selectively secure.*

Proof. Take any bit $b \in \{0,1\}$. Suppose there exists an efficient adversary $\mathcal{A}$ that breaks selective security of Construction 7.2 with non-negligible advantage ε. We use $\mathcal{A}$ to construct a new adversary $\mathcal{B}$ for the N-BDHE problem as follows:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives the N-BDHE challenge
$$(h, g, g_1, \ldots, g_N, g_{N+2}, \ldots, g_{2N}, T).$$
2. Adversary $\mathcal{B}$ starts running $\mathcal{A}$ to receive a set $S \subseteq [N]$.
3. Adversary $\mathcal{B}$ samples $\rho \xleftarrow{R} \mathbb{Z}_p$. It then computes $v = \frac{g^\rho}{\prod_{j \in S} g_{N+1-j}}$, and sets the master public key to be
$$\mathsf{mpk} = \big(g, g_1, g_2, \ldots, g_N, g_{N+2}, \ldots, g_{2N}, v\big).$$
Next, for each $i \notin S$, the reduction algorithm computes $d_i = \frac{g_i^\rho}{\prod_{j \in S} g_{N+1-j+i}}$ and sets $\mathsf{sk}_i = (i, g_i, d_i)$. It sends mpk and $\{\mathsf{sk}_i\}_{i \notin S}$ to $\mathcal{A}$.
4. After $\mathcal{A}$ outputs a pair of messages m_0, m_1, algorithm $\mathcal{B}$ sets $\mathsf{ct}_1^* = h$, $\mathsf{ct}_2^* = h^\rho$, and $\mathsf{ct}_3^* = T \cdot m_b$ and gives $\mathsf{ct}^* = (\mathsf{ct}_1^*, \mathsf{ct}_2^*, \mathsf{ct}_3^*)$ to $\mathcal{A}$.
5. After $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, algorithm $\mathcal{B}$ outputs the same bit.

From our earlier analysis, we conclude that algorithm $\mathcal{B}$ perfectly simulates the selective security game with bit b whenever $T = e(g,g)^{\alpha^{N+1}\beta}$ and the selective security game with a message-independent ciphertext (i.e., the distribution in Eq. (8.2)) if $T = e(g,g)^t$. Correspondingly, algorithm $\mathcal{B}$ breaks the N-BDHE assumption with the same advantage. □

Remark 8.4 (Complexity Leveraging). Why can't we just guess S beforehand and prove adaptive security? This is an idea called complexity leveraging. In a setting like IBE, selective security implies adaptive security by this very approach, because you can guess such an element correctly with $\frac{1}{2^\lambda}$, meaning the proof is still sound due to subexponential hardness. However this does not hold in broadcast encryption because the probability would be reduced by $\frac{1}{2^N}$. This could cause the reduction to become negligible, so we would have to show that the algorithm does not allow the adversary to succeed even with $\epsilon/2^N$. This means we would have to scale up the security parameter by at least N, to $\lambda + N$. The issue now is that the size of the ciphertext grows, so it is no longer succinct. Recall that we can trivially implement broadcast encryption with a linear size ciphertext.

Limitation of broadcast encryption. Another name for this construction of broadcast encryption is Centralized Broadcast Encryption, because broadcast encryption changes the model familiar to us in public-key encryption, in the sense that there is now a setup algorithm that generates $\mathsf{mpk}, \mathsf{sk}_1, \mathsf{sk}_2, \cdots$. In order to run this setup algorithm, there must now be a central point of failure, where the central authority that runs setup needs to be trusted. Instead of giving power to the end user, we have concentrated power in a single entity, which some argue is a step backwards in the field of cryptography.

Distributed broadcast encryption. DBE is a notion that aims to solve the problem of having a central point of failure. In ths model, there is a public key directory similar to a giant bulletin board. Users can generate their own secret keys (and public keys), then post their public keys in the directory. It is trivial to see that broadcast encryption can still be done in linear ciphertext size; however, we would like to achieve broadcast encryption with sublinear ciphertext size. Specifically, we would like to broadcast to $\{pk_i\}_{i\in S}$ with ciphertext ct of size $|\mathsf{ct}| = o(|S|)$. In fact, such a scheme was discovered a few months ago, which uses the BGW construction to derive a distributed broadcast encryption construction.

Definition 8.5 (Distributed Broadcast Encryption)**.** A distributed broadcast encryption scheme is an encryption scheme with the following algorithms:

- $\mathsf{Setup}() \to \mathsf{pp}$: Generate some public parameters for the scheme to use.
- $\mathsf{KeyGen}(\mathsf{pp}, i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$: Run by an individual user i, generates a keypair for the user based on the public parameters.
- $\mathsf{Encrypt}(\mathsf{pp}, (i, \mathsf{pk}_i)_{i\in S}, m) \to \mathsf{ct}$: Given a set of users and their public keys, as well as the public parameters, encrypt a message to produce a ciphertext which should only be readable by users in S.
- $\mathsf{Decrypt}((i, \mathsf{pk}_i)_{i\in S}, \mathsf{sk}_i, \mathsf{ct}) \to m$: Given a ciphertext, as well as the set of users it was encrypted for along with all their public keys, use the individual user's secret key to decrypt the ciuphertext, returning the broadcasted message m.

Construction 8.6 (Kolonelos-Malavolta-Wee Distributed Broadcast Encryption Scheme [KMW23])**.** This construction is similar to the BGW scheme, but instead of having a common γ, each user needs to choose their own γ as all secrets must be introduced during KeyGen.

- $\mathsf{Setup}(n)$: Sample $\alpha \xleftarrow{R} \mathbb{Z}_p$. Output $\mathsf{pp} = (g, g_1, g_2, \cdots, g_N, g_{N+2}, \cdots, g_{2N})$.
- $\mathsf{KeyGen}(\mathsf{pp}, i)$: Parse $\mathsf{pp} = (g, g_1, g_2, \cdots, g_N, g_{N+2}, \cdots, g_{2N})$. Sample $\gamma_i \xleftarrow{R} \mathbb{Z}_p$. Generate the user's secret key $\mathsf{sk}_i = (g_i, g_i^{\gamma})$ and public key $\mathsf{pk}_i = (g^{\gamma_i}, g_1^{\gamma_i}, g_2^{\gamma_i}, \cdots, g_{i-1}^{\gamma_i}, g_{i+1}^{\gamma_i}, \cdots, g_N^{\gamma_i})$. Output $(\mathsf{sk}_i, \mathsf{pk}_i)$.
- $\mathsf{Encrypt}(\mathsf{pp}, (i, \mathsf{pk}_i)_{i\in S}, m)$: Parse $\mathsf{pp} = (g, g_1, g_2, \cdots, g_N, g_{N+2}, \cdots, g_{2N})$, as well as $(i, \mathsf{pk}_i)_{i\in S} = (i, g^{\gamma_i})_{i\in S}$. Compute the product $v_S = \prod_{i\in S} g^{\gamma_i}$. Output the ciphertext $\mathsf{ct} = (g^r, (v_S \prod_{j\in S} g_{N+1-j})^r, e(g, g_N)^r m)$.
- $\mathsf{Decrypt}((i, \mathsf{pk}_i)_{i\in S}, \mathsf{sk}_i, \mathsf{ct})$: Parse $(i, \mathsf{pk}_i)_{i\in S} = (i, g^{\gamma_i})_{i\in S}$, parse $\mathsf{sk}_i = (g_i, g_i^{\gamma})$, as well as $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3)$. Output $\mathsf{ct}_3^{-1} \frac{e(g_i, \mathsf{ct}_2)/e(g_i^{\gamma_i} \prod_{j\in S\setminus i} g_{N+1-j+i}, \mathsf{ct}_1)}{\prod_{j\in S\setminus i} e(g_i^{\gamma_j}, \mathsf{ct}_1)}$.

Derivation. The following identities inform our decision for the Decrypt algorithm:

$$
\begin{aligned}
&e(g_i, (v_S \prod_{j \in S} g_{N+1-j})^r) \\
&= e(g_i, v_S)^r e(g_1, g_N)^r \prod_{j \in S \setminus \{i\}} e(g, g_{N+1-j+i})^r \\
&= e(g_i, \prod_{j \in S} g^{\gamma_i})^r e(g_1, g_N)^r \prod_{j \in S \setminus \{i\}} e(g, g_{N+1-j+i})^r \\
&= e(g_i^{\gamma_i}, g^r) \prod_{j \in S \setminus i} e(g_i, g^{\gamma_j})^r e(g_1, g_N)^r \prod_{j \in S \setminus \{i\}} e(g, g_{N+1-j+i})^r \\
&= e(g_i^{\gamma_i}, g^r) \prod_{j \in S \setminus i} e(g, g_i^{\gamma_j})^r e(g_1, g_N)^r \prod_{j \in S \setminus \{i\}} e(g, g_{N+1-j+i})^r
\end{aligned}
$$

Remark 8.7. Unlike in the BGW scheme, individual public keys now scale with N. This is because we need to provide hints through each public key so that Decrypt for user i can successfully cancel out terms which are dependent on user $j \neq i$'s key.

Remark 8.8. The Setup algorithm still needs to be run by a trusted party which will not remember α after the fact. In practice, such as for cryptocurrencies, this is usually done through a multiparty computation during a physical ceremony. Afterwards, the hard drive which the algorithm was run on will be destroyed.

CS 395T: Topics in Cryptography February 13, 2024

Lecture 9: Attribute-Based Encryption

Lecturer: David Wu *Scribe: Phuc Dang*

In an identity-based encryption, both secret keys and ciphertexts are associated with identities, and decryption is successful if the identities match. In this setting, the decryption policy can be viewed as testing for equality (of the identities associated with the secret key and the ciphertext). In this lecture, we study a generalization of identity-based encryption to *attribute-based encryption* (ABE) [SW05, GPSW06] which supports a broader class of decryption policies. For instance, in key-policy ABE, the ciphertext is associated with a set of attributes S while the decryption keys are associated with a policy P. Decryption succeeds whenever the set of attributes satisfy the policy (i.e., we can denote this by writing $P(S) = 1$).

As an example, an encrypter might tag ciphertexts with a classification level (e.g., UNCLASSIFIED, SECRET, or TOP SECRET) and the decryption policy associated with a key could be to allow decryption of all ciphertexts that have the UNCLASSIFIED or SECRET attribute, but not ciphertexts with a TOP SECRET attribute. In this way, attribute-based encryption provides *fine-grained* access control to encrypted data. The access policies are encoded in the semantics of the encryption scheme itself.

Definition 9.1 (Attribute-Based Encryption)**.** Let $\mathcal{U}$ be the universe of possible attributes and $\mathcal{P}$ be a family of policies over $\mathcal{U}$. We write $2^{\mathcal{U}}$ to denote the power set of $\mathcal{U}$ (i.e., the set of all subsets of $\mathcal{U}$). Then each policy $P \in \mathcal{P}$ is a mapping $P\colon 2^{\mathcal{U}} \to \{0,1\}$ that takes as input a set of attributes $S \subseteq \mathcal{U}$ and outputs a bit $b \in \{0,1\}$ indicating whether the set of attributes satisfy the policy ($b = 1$) or not ($b = 0$). A (key-policy) attribute-based encryption scheme with attribute universe $\mathcal{U}$, policy family $\mathcal{P}$ and message space $\mathcal{M}$ is a tuple of efficient algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{msk}, \mathsf{mpk})$: On input the security parameter λ, the setup setup algorithm outputs the master public key mpk and the master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, P)$: On input the master secret key msk and the policy $P \in \mathcal{P}$, the key-generation algorithm outputs a secret key sk_P associated with this policy.
- $\mathsf{Encrypt}(\mathsf{mpk}, S, m) \to \mathsf{ct}$: On input the master public key mpk, a set of attributes $S \subseteq \mathcal{U}$, and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to m$: On receiving a secret key sk and a ciphertext ct, the decryption algorithm either outputs a message $m \in \mathcal{M}$ or a special symbol $m = \perp$ (to indicate decryption failure).

Correctness. The scheme is correct if for some given some policy P and all sets $\mathsf{S} \subseteq \mathbb{U}$ such that $\mathsf{P}(\mathsf{S}) = 1$, then $\mathsf{Pr}[\mathsf{Decrypt}(\mathsf{sk}_\mathsf{P}, \mathsf{ct}) = \mathsf{m}] = 1$.

Security. We now define semantic security for an ABE scheme. Security says that the adversary cannot learn anything about a message encrypted to the set of attributes S if it does not have a secret key sk_P with a corresponding policy P where $\mathsf{P}(\mathsf{S}) = 1$. We will now define this formally. As usual, the game is parameterized by a bit $b \in \{0,1\}$.

- **Setup:** At the beginning of the game, the challenger samples $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}()$ and sends mpk to the adversary $\mathcal{A}$.

- **Key-generation queries:** The adversary $\mathcal{A}$ is now allowed to issue key-generation queries on policies P of its choosing. On each query, the challenger responds with the secret key $\mathsf{sk}_{\mathsf{P}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{P})$.
- **Challenge query:** When the adversary is done making key-generation queries, it outputs a challenge set S^* and two messages (m_0^*, m_1^*). The challenger samples a bit $b \xleftarrow{R} \{0,1\}$ and responds with $\mathsf{ct}_b^* \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, \mathsf{S}^*, m_b^*)$
- **Key-generation queries:** The challenger can continue to make key-generation queries on identities of its choosing. The challenger responds to each query exactly as before.
- **Output:** At the end of the experiment, the adversary outputs a bit $b' \in \{0,1\}$.

We say that adversary $\mathcal{A}$ is admissible for the semantic security game if it does not make a key-generation query on the challenge set S^*. Then, we say that the ABE scheme is semantically secure if for every efficient adversary $\mathcal{A}$,

$$|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| \leq \mathsf{negl}(\lambda).$$

Goyal-Pandey-Sahai-Waters (GPSW). We now introduce the ABE scheme by Goyal et al [GPSW06]. It uses an El-Gamal style encryption. We will build this scheme firstly from a general construction. Then, we will build a visualization of what a policy tree is and how it makes sense of the boolean formulas used in generating the share matrix.

Construction 9.2. Assume that the number of attributes (or alternatively, the universe $\mathbb{U}$) is polynomial, labeling them by integers $1, 2, ..., n$.

- **Setup:** For each $i \in [n]$, sample $t_i \xleftarrow{R} \mathbb{Z}_p$. Sample $\gamma \xleftarrow{R} \mathbb{Z}_p$. Then, output the attribute keys $(T_1 = g^{t_1}, ..., T_n = g^{t_n}, h = e(g,g)^{\gamma})$. Let $\mathsf{mpk} = (T_1, ..., T_n, h)$ and $\mathsf{msk} = (t_1, ..., t_n, \gamma)$.
- **Encryption:** To encrypt a message $m \in \mathbb{G}_T$ using the mpk and a set of attributes $\mathsf{S} \subseteq [n]$, sample $r \xleftarrow{R} \mathbb{Z}_p$ and output $ct = (\mathsf{S}, \{T_i^r\}_{i \in \mathsf{S}}, h^r * m)$.
- **Key Generation:** Given the msk, generating a key for policy P is as follows:
  1. Secret share γ according to policy P.
  2. Let $\gamma_i \in \mathbb{Z}_p$ be the share associated with attribute i.
  3. The secret key for P is the set g^{γ_i / t_i} for each attribute i that appears in the policy along with policy P

  There are still a few details in this key generation algorithm to go over. Namely, how one should secret share γ to the policy P. For this, we'll need to describe the policy as a boolean formula. A policy can be visualized as a tree of AND gates and OR gates. For example, say we have a secret key sk_{P} according to some policy P. This policy in tree form looks like this:

  The leaves 1, 2, 3, 4, and 5 are attributes that a message encrypted to some set S must fulfill. More concretely, if a message was encrypted to $\mathsf{S} = (1, 2)$, this secret key with the corresponding policy P visualized above would be able to decrypt it. However, if the message was encrypted to $\mathsf{S} = (1, 4)$, then this secret key would not be able to decrypt.

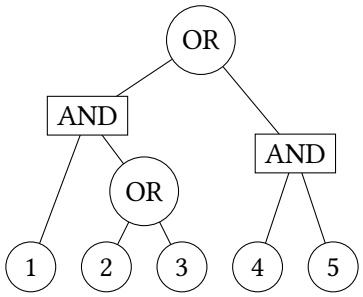
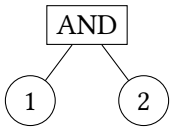
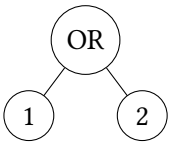


Figure 9.1: An example policy tree. Can also be represented as $(1 \wedge (2 \vee 3)) \vee (4 \wedge 5)$.

This is hard to digest how exactly secret sharing works with this policy tree, so breaking it down further, let's describe how γ is secret shared if the policy had just one gate:



(a) Sample $\gamma_1, \gamma_2 \xleftarrow{R} \mathbb{Z}_p$ such that $\gamma = \gamma_1 + \gamma_2$. This implies that we need both γ_1 AND γ_2 to obtain γ. The resulting key is $sk_{\mathsf{P}} = (g^{\gamma_1/t_1}, g^{\gamma_2/t_2})$.



(b) Propagate the value γ to 1 and 2 such that $\gamma_1 = \gamma_2 = \gamma$. This implies that we only need either γ_1 OR γ_2 to obtain γ. The resulting key is $sk_{\mathsf{P}} = (g^{\gamma/t_1}, g^{\gamma/t_2})$.

To simplify, sk_{P} can be referred to as $K = (K_1, ..., K_m) = (g^{\gamma_1/t_1}, ..., g^{\gamma_m/t_m})$ where m denotes the number of leaves in the policy tree.

- **Decryption:** Given $ct = (\mathsf{P}, \{T_i^r\}_{i \in \mathsf{S}}, h^r * m)$ and sk_{P}, decryption succeeds if the attributes of set S satisfies the boolean formula specified by the policy of our secret key.

  For example, let our policy associated with our secret key be the one defined by the example tree [9.1] $K = (K_1, K_2, K_3, K_4, K_5)$, and let the ciphertext have the attribute set $\mathsf{S} = (2, 4, 5)$. This means that this part $\{T_i^r\}_{i \in \mathsf{S}}$ of the ciphertext contains $(g^{rt_2}, g^{rt_4}, g^{rt_5})$ which will be instrumental in constructing the h^r term needed to retrieve m.

  Pairing these terms $e(K_i, T_i^r)$ for $i \in \mathsf{S}$, gets you $e(g^{\gamma_i/t_i}, g^{rt_i}) = e(g, g)^{\gamma_i r}$. Notice here that the exponents of these results are secret shares of γr. More specifically, these secret shares are derived from a linear secret sharing scheme. With our example policy tree [9.1], we can use the result of $e(g, g)^{\gamma_4 r}$ and $e(g, g)^{\gamma_5 r}$ to reconstruct $e(g, g)^{\gamma r}$.

  The product of $e(g, g)^{\gamma_4 r} * e(g, g)^{\gamma_5 r}$ is $e(g, g)^{(\gamma_4 + \gamma_5) r}$. The exponent $(\gamma_4 + \gamma_5) r$ indeed perfectly reconstructs γr. Note then that we have constructed $h^r = e(g, g)^{\gamma r} = e(g, g)^{(\gamma_4 + \gamma_5) r}$. Decryption is thus trivial.

Remark 9.3. The last thing for this construction is to show how to create a share generating matrix from a boolean formula.

- **AND gate:** For an AND gate, we can secret share γ by sampling $\alpha \xleftarrow{R} \mathbb{F}_p$ and setting $\gamma_1 = \gamma + \alpha$ and $\gamma_2 = -\alpha$:

$$\begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} \gamma \\ \alpha \end{bmatrix}$$

- **OR gate:** For an OR gate, we can secret share γ by setting $\gamma_1 = \gamma_2 = \gamma$:

$$\begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \gamma \end{bmatrix}$$

- **General Procedure:** To associate a policy tree with a share generating matrix:

  1. Associate vector $[1]$ with root node and initialize counter $c \leftarrow 1$.
  2. Proceed down the tree.
     - If a node is an OR with label v, the children are associated with label v (i.e. they are identical to the parent).
     - If a node is an AND with label v, we label the children with label $v \| 0^{len(v)-c} \| 1$ and the other label with $0^c \| -1$. Observe that the sum of the shares is $v \| 0^{len(v)-c}$. Finally, increment $c \leftarrow c + 1$.
  3. Pad all vectors with 0s to dimension c.

  For example, using the policy tree we've introduced above [9.1], we will label it accordingly to these steps.

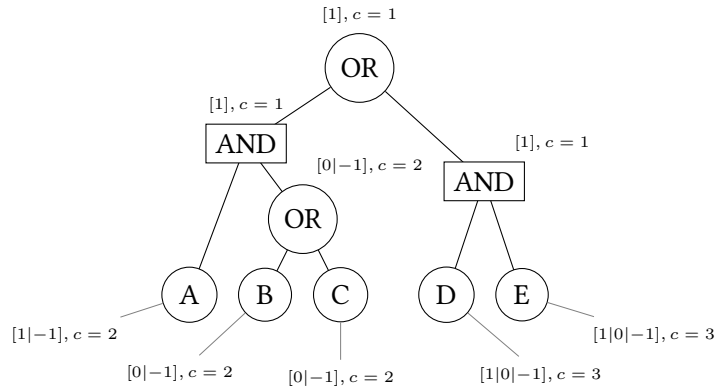


Figure 9.3: A labeled policy tree, now with alphabetic leaf labeling.

The share-generation matrix follows suit from this tree, where the first row corresponds to the share held by A, the second corresponds to the share held by B and so on:

$$\underbrace{\begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & -1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix}}_{M_{\mathsf{P}}} \cdot \begin{bmatrix} \gamma \\ \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} \gamma + \alpha_1 \\ -\alpha_1 \\ -\alpha_1 \\ \gamma + \alpha_2 \\ -\alpha_2 \end{bmatrix}$$

- **Main observation:** If the attributes of the ciphertext satisfy policy, then the vector $e_1^\intercal = [1, 0, ..., 0]$ will be in the row span of M_p. Conversely, if the attributes do not satisfy policy, then the vector $e_1^\intercal$ is not in the row span of M_p. More generally, the boolean formula policy P on k attributes can be transformed into a share generating matrix M with k rows.

Construction 9.4. We will now reconstruct the GPSW scheme with this share generating matrix M in mind. Let n denote the attribute universe.

- $\mathsf{Setup}(n) \to (\mathsf{msk}, \mathsf{mpk})$: Sample $t_1, ..., t_n \xleftarrow{R} \mathbb{Z}_p$ and $\gamma \xleftarrow{R} \mathbb{Z}_p$. Set $\mathsf{mpk} = (T_1, ..., T_n, h)$ and $\mathsf{msk} = (t_1, ..., t_n, \gamma)$ where $T_i = g^{t_i}$ and $h = e(g, g)^\gamma$.
- $\mathsf{Encrypt}(\mathsf{mpk}, \mathsf{S}, m) \to \mathsf{ct}$: Sample $r \xleftarrow{R} \mathbb{Z}_p$ and set $ct = (\{(i, T_i^r)\}_{i \in \mathsf{S}}, h^r m)$.
- $\mathsf{KeyGen}(\mathsf{msk}, M) \to \mathsf{sk}$: As shown before, $M \in \mathbb{Z}_p^{k \times l}$ is the share generating matrix associated with the policy. Sample $\alpha_1, ..., \alpha_{l-1} \xleftarrow{R} \mathbb{Z}_p$ and let $N = [\gamma, \alpha_1, \alpha_2, ..., \alpha_{l-1}]$. Compute the shares of λ:

$$\begin{bmatrix} s_1 \\ \vdots \\ s_k \end{bmatrix} = M \cdot \begin{bmatrix} \gamma \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{l-1} \end{bmatrix} = M \cdot N^\mathsf{T}$$

  Then, let $\mathsf{p} : [k] \to [n]$ be the mapping from the row index of M to the attribute index in $[n]$. Output $\mathsf{sk} = (M, \{(i, g^{s_i / t_{\mathsf{p}(i)}})\}_{i \in [k]})$.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to m$: Parse $\mathsf{sk} = (M, \{(i, D_i)\}_{i \in [k]})$ and $\mathsf{ct} = (\{(i, C_i)\}_{i \in \mathsf{S}}, \mathsf{Z})$.

  If S satisfies the policy, then there exists a vector $v \in \mathbb{Z}_p^k$ such that $v^\mathsf{T} M = [1, 0, ..., 0]$ and $v_i = 0$ for all $\mathsf{p}(i) \in \mathsf{S}$.

  Compute $\dfrac{\mathsf{Z}}{\prod_{i \in [t]} e(C_{\mathsf{p}(i)}, D_i)^{v_i}}$.

Correctness. We can prove correctness as follows:

$$\begin{aligned} \prod_{i \in [t]} e(C_{\mathsf{p}(i)}, D_i)^{v_i} &= \prod_{i \in [t]} e(g^{r t_{\mathsf{p}(i)}}, g^{s_i / t_{\mathsf{p}(i)}})^{v_i} \\ &= \prod_{i \in [t]} e(g, g)^{r s_i v_i} \\ &= e(g, g)^{r v^\mathsf{T} M \cdot N^\mathsf{T}} = e(g, g)^{\gamma r} = h^r \end{aligned}$$

Security. Security proof to follow in the next lecture...

CS 395T: Topics in Cryptography February 15, 2024

Lecture 10: Attribute-Based Encryption

Lecturer: David Wu *Scribe: Shafik Nassar*

Last Lecture Review. In the last lecture, we started discussing the Goyal-Pandey-Sahai-Waters (GPSW) Attribute-Based Encryption (ABE) scheme. Recall that the GPSW scheme is a key-policy ABE, meaning that each secret key contains a policy and each message is encrypted to a certain set of attributes which are labeled by integers $1, \ldots, n$. The correctness requirement states that a user with access policy $P : \{0,1\}^n \to \{0,1\}$ should be able to decrypt a ciphertext that was encrypted to a set $S \in \{0,1\}^n$ (a subset of $[n]$ can be represented by an n bit string) if $P(S) = 1$.

We also saw that a Boolean access formula over k attributes can be associated with a $k \times \ell$ *share-generating matrix* M, where ℓ is the number of AND gates in the formula. To share a secret $\gamma \in \mathbb{Z}_p$ using M, we first sample uniform $\alpha_1, \ldots, \alpha_{\ell-1} \xleftarrow{R} \mathbb{Z}_p$ and compute:

$$M \cdot \begin{bmatrix} \gamma \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{\ell-1} \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_k \end{bmatrix}$$

The matrix M satisfies the following: if a set $S \subseteq [k]$ satisfies the access policy then there exists a vector $v \in \mathbb{Z}_p^k$ such that

$$v^\intercal \cdot M = [1\ 0\ 0 \ldots 0] \text{ and } \forall i \notin S : v_i = 0.$$

The first condition allows us to reconstruct the secret, and the second condition guarantees that we only need the attributes in the set S to do so. We can see that by examining the following equation:

$$\gamma = v^\intercal \cdot M \cdot \begin{bmatrix} \gamma \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{\ell-1} \end{bmatrix} = v^\intercal \cdot \left(M \cdot \begin{bmatrix} \gamma \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{\ell-1} \end{bmatrix} \right)$$

Secret sharing security. The previous property guarantees correctness. Additionally, we want to guarantee security. Consider the following property. If $S \subseteq [k]$ does not satisfy the access policy then the vector $e_1 = [1\ 0\ 0 \ldots 0]$ is *not* spanned by the rows of M that are associated with S (rows i such that $i \in S$). Clearly, this property is necessary for security: if the vector e_1 is spanned by the rows associated with S, then we can simply take the linear combination that gives us e_1, and reconstruct γ directly. We also claim that this property is sufficient for security. Suppose $v_1, \ldots, v_m \in \mathbb{Z}_p^\ell$ for $m < \ell$ and suppose that $e_1 \notin \mathsf{span}(v_1, \ldots, v_m)$. Then e_1 decomposes to two parts $e_1 = e_{1,V} + e_{1,V^\perp}$, such that $e_{1,V}$ is in $\mathsf{span}(v_1, \ldots, v_m)$ and $e_{1,V^\perp}$ is in the subspace that is orthogonal to $\mathsf{span}(v_1, \ldots, v_m)$. This implies that there exists a $w \in \mathbb{Z}_p$ such that $\forall i \in [n] : w^\intercal v_i = 0$ and $w^\intercal e_1 \neq 0$, and without loss of generality, the first coordinate of w can be 1. This implies that the secret γ can never be "isolated" using the vectors $v_1, \ldots, v_m$, therefore we have perfect secrecy. We will come back to this property later on when we prove the security of GPSW.

GPSW ABE scheme. Recall the GPSW scheme:

- $\mathsf{Setup}(n) \to (\mathsf{msk}, \mathsf{mpk})$:
  1. Sample $t_1, ..., t_n \xleftarrow{R} \mathbb{Z}_p$ and set $T_i = g^{t_i}$ for all $i \in [n]$.
  2. Sample $\gamma \xleftarrow{R} \mathbb{Z}_p$ and set $h = e(g,g)^\gamma$.
  3. Output $\mathsf{mpk} = (T_1, ..., T_n, h)$ and $\mathsf{msk} = (t_1, ..., t_n, \gamma)$.
- $\mathsf{Encrypt}(\mathsf{mpk}, S, m) \to \mathsf{ct}$: Sample $r \xleftarrow{R} \mathbb{Z}_p$ and set $ct = (\{(i, T_i^r)\}_{i \in S}, h^r m)$.
- $\mathsf{KeyGen}(\mathsf{msk}, M) \to \mathsf{sk}$: The matrix $M \in \mathbb{Z}_p^{k \times \ell}$ is the share generating matrix, and k is the number of attributes that M depends on.
  1. Sample $\alpha_1, \ldots, \alpha_{\ell-1} \xleftarrow{R} \mathbb{Z}_p$, and secret share γ:
  $$\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_k \end{bmatrix} \leftarrow M \cdot \begin{bmatrix} \gamma \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{\ell-1} \end{bmatrix}$$
  2. Let $\rho : [k] \to [n]$ be a share labeling function that translates an index in the rows of M to an attribute. We can skip this, but then the size of M (and thus the secret keys) will grow with n instead of k.
  3. Output $\mathsf{sk} = \left(M, \left\{\left(i, g^{s_i / t_{\rho(i)}}\right)\right\}_{i \in [k]}\right)$.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to m$: Parse $\mathsf{sk} = (M, \{(i, D_i)\}_{i \in [k]})$ and $\mathsf{ct} = (\{(i, C_i)\}_{i \in S}, z)$. If S does not satisfy the policy, output $\perp$.Otherwise, there exists a vector $v \in \mathbb{Z}_p^k$ such that $v^\intercal M = [1, 0, ..., 0]$ and $v_i = 0$ for all $\rho(i) \in S$. Output
$$\frac{z}{\prod_{i \in [k]} e(C_{\rho(i)}, D_i)^{v_i}}.$$

Correctness was proved in the previous lecture. We move on to security.

Selective Security. In the selective security game, the adversary commits to a set $S^* \subseteq [n]$ ahead of time. We actually show a stronger property than indistinguishability. We prove that the ciphertext is indistinguishable from a distribution that does not depend on the message (almost that the ciphertexts are pseudorandom). That is

$$\left(\{(i, T_i^r)\}_{i \in S}, h^r m\right) \approx \left(\{(i, T_i^r)\}_{i \in S}, e(g,g)^\alpha \text{ where } \alpha \xleftarrow{R} \mathbb{Z}_p\right).$$

We explicitly state the security game, which is parameterized by a bit $b \in \{0, 1\}$.

1. **Set declaration:** The adversary $\mathcal{A}$ sends the number of attributes 1^n and a set $S^* \subseteq [n]$.
2. **Setup:** The challenger samples $\mathsf{msk} \leftarrow (t_1, \ldots, t_n, \gamma)$ and $\mathsf{mpk} \leftarrow (T_1, \ldots, T_n, h)$ as in Setup and sends mpk to $\mathcal{A}$.

3. **Key-generation queries:** The adversary $\mathcal{A}$ is now allowed to issue key-generation queries on access policy matrices $M \in \mathbb{Z}_p^{k \times \ell}$ of its choosing. On each query, the challenger responds with the secret key $\mathsf{sk}_M \leftarrow (M, \{(i, D_i)\}_{i \in [k]})$ which is computed as in KeyGen.

4. **Challenge query:** When the adversary is done making key-generation queries, it outputs a message m^*. If $b = 0$ then the challenger computes ct^* as an encryption of m^*, that is $\mathsf{ct}^* \leftarrow (\{(i, T_i^r)\}_{i \in S^*}, h^r m^*)$ where $r \xleftarrow{R} \mathbb{Z}_p$. If $b = 1$ then the challenger samples additionally a random $\alpha \xleftarrow{R} \mathbb{Z}_p$ and sets $\mathsf{ct}^* \leftarrow (\{(i, T_i^r)\}_{i \in S^*}, e(g,g)^\alpha)$. The challenger sends ct^* to $\mathcal{A}$.

5. **Key-generation queries:** The challenger can continue to make key-generation queries on identities of its choosing. The challenger responds to each query exactly as before.

6. **Output:** At the end of the experiment, the adversary outputs a bit $b' \in \{0, 1\}$.

Security reduction. Recall the Decisional Bilinear Diffie Hellman (DBDH) assumption:

$$\left(g, g^a, g^b, g^c, e(g,g)^{abc}\right) \overset{c}{\approx} \left(g, g^a, g^b, g^c, e(g,g)^d\right),$$

where $a, b, c, d \xleftarrow{R} \mathbb{Z}_p$. The challenging part is that the reduction algorithm needs to generate secret keys for all policies except the ones that are able to decrypt (ciphertexts for) S^*. Recall that this is a good sanity check: if the reduction can beat the security game without the help of the adversary then there is something wrong. This is similar to the case of identity based encryption, but it is harder for policies!

Assume $\mathcal{A}$ is an adversary for the selective security game. Here is a reduction strategy to DBDH:

1. The reduction receives the tuple (g, A, B, C, U).

2. Adversary $\mathcal{A}$ declares set $S^* \subseteq [n]$.

3. The reduction algorithm needs to do the following:

   (a) Construct mpk, namely construct $T_1, \ldots, T_n$ and $h = e(g,g)^\gamma$.

   (b) Construct ct^*, namely construct $\{T_i^r\}_{i \in S^*}$ and the blinding factor $e(g,g)^{\gamma r}$. Note that if $b = 1$ in the experiment then the blinding factor is a random element in the target group.

   (c) Answer key generation queries. This is the hardest part.

4. Eventually, when $\mathcal{A}$ makes a guess, the reduction outputs that same guess.

The reduction needs to map γ and r to a, b, c. Let us use $r \leftarrow c$ and $\gamma \leftarrow ab$.

$$h \leftarrow e(A, B) = e(g,g)^{ab} = e(g,g)^\gamma. \tag{10.1}$$

Since the reduction wants to use the guess of $\mathcal{A}$ regarding the encryption in order to distinguish between the case where $U = e(g,g)^{abc}$ and U is a random element in the target group, then it better be the case that the blinding factor $e(g,g)^{\gamma r}$ that is constructed in the ciphertext is U. This means that the reduction does not know r, therefore it must know t_i for all $i \in S^*$.

$$\text{For all } i \in S^* : t_i \xleftarrow{R} \mathbb{Z}_p \text{ and set } T_i \leftarrow g^{t_i}. \tag{10.2}$$

$$\text{For all } i \in S^* : T_i \leftarrow C^{t_i} = T_i^r. \tag{10.3}$$

We still need to choose T_i for $i \notin S^*$. We will do that as we figure out how answer key-generation queries.

Answering key-generation queries. Let $M \in \mathbb{Z}_p^{k \times \ell}$ be the policy (matrix) that $\mathcal{A}$ queries. Denote row i of M by $m_i^\intercal$. Let $\rho : [k] \to [n]$ be the index translation function. Normally, the ABE challenger would sample uniform $\alpha_1, \ldots, \alpha_{\ell-1}$, set $f = [ab\ \alpha_1\ \ldots\ \alpha_{\ell-1}]^\intercal$ and compute the shares $s_1, \ldots, s_k$ using $M \cdot f$. Finally, the secret key would consist of $(M, \{(i, D_i)\}_{i \in [k]})$, where $D_i = g^{s_i / t_{\rho(i)}}$. Our reduction will have to construct D_i differently. First, we can decompose f as follows

$$f = b \cdot z + ab \cdot w \text{ where } w = \begin{bmatrix} 1 \\ w_2 \\ \vdots \\ w_\ell \end{bmatrix} \text{ and } z = \begin{bmatrix} 0 \\ \alpha_1 \\ \vdots \\ \alpha_{\ell-1} \end{bmatrix}$$

Specifically, because we know that the queried policy M is not satisfied by S^*, we can choose w, as in the discussion about the secret sharing scheme, to be orthogonal to $\{m_i^\intercal\}_{i \in S^*}$ (the rows of M that correspond to S^*). Now we split to two cases. The easiest one is if $\rho(i) \in S^*$. In this case, we know $t_{\rho(i)}$ and additionally have $m_i^\intercal w = 0$. This last cancellation is what saves us. This allows us to simplify the exponent:

$$\frac{s_i}{t_{\rho(i)}} = \frac{bm_i^\intercal z + abm_i^\intercal w}{t_{\rho(i)}} = \frac{bm_i^\intercal z}{t_{\rho(i)}}.$$

The reduction can compute $\frac{m_i^\intercal z}{t_{\rho(i)}}$, and $g^b = B$, therefore the reduction can compute $D_i = g^{s_i / t_{\rho(i)}} = B^{\frac{m_i^\intercal z}{t_{\rho(i)}}}$. If $\rho(i) \notin S^*$, then the problem here is that we do not have $t_{\rho(i)}$. To solves the problem, we choose $t_{\rho(i)} = \delta_i \cdot b$ for some random δ_i. This preserves the distribution of $t_{\rho(i)}$. The exponent can be simplified again

$$\frac{s_i}{t_{\rho(i)}} = \frac{bm_i^\intercal z + abm_i^\intercal w}{t_{\rho(i)}} = \frac{m_i^\intercal z + am_i^\intercal w}{\delta_i}.$$

The reduction can compute $\frac{m_i^\intercal z}{\delta_i}$ and $\frac{m_i^\intercal w}{\delta_i}$. Again, g^a is simply A. This allows the reduction to compute $D_i = g^{\frac{m_i^\intercal z}{\delta_i}} A^{\frac{m_i^\intercal w}{\delta_i}}$. Note that by setting $t_{\rho(i)}$ for $i \notin S^*$ we have also set the corresponding $T_{\rho(i)} = g^{b\delta_i}$, and thus we have completed the construction of mpk. In total, the reduction can perfectly simulate the security game for $\mathcal{A}$ and answers 0 if $U = e(g, g)^{abc}$ and answers 1 if $U = e(g, g)^d$.
Note that we have implicitly used the perfect secrecy of the secret sharing scheme when constructing D_i.

CS 395T: Topics in Cryptography February 20, 2024

Lecture 11: Somewhat Homomorphic Encryption

Lecturer: David Wu *Scribe: Zhiyi Huang*

In the previous lectures, we mainly focused on topics of different applications of pairing-based encryption schemes. From now on we will turn to the topics of proof systems. We will start by a question: Can we compute on encrypted data? This will lead to the idea of Homomorphic Encryption. While Fully Homomorphic Encryption(FHE) is allowing us to do arbitrary computation on ciphertexts using only the public key, we will start by a simpler version which only allows us to do limited computation on the ciphertext.

Additively homomorphic encryption. We start by describing a variant of classic ElGamal encryption which is additively homomorphic:

- $\mathsf{Setup}() \to \mathsf{pk}, \mathsf{sk}$: We sample $x \xleftarrow{R} \mathbb{Z}_p$ and set $\mathsf{pk} = (g, h = g^x)$, $\mathsf{sk} = x$.
- $\mathsf{Encrypt}(\mathsf{pk}, m) \to \mathsf{ct}$: Sample $r \xleftarrow{R} \mathbb{Z}_p$ and set $\mathsf{ct} = (g^r, u = h^r \cdot m)$.
- $\mathsf{Decrypt}(\mathsf{sk}, ct) \to m$: Output $m = \frac{u}{(g^r)^x}$

In order to support addition, we need to modify this scheme a little bit: where we set the ciphertext $\mathsf{ct} = (g^r, u = h^r \cdot g^m)$, where we assume the message $m \in Z$. However, now in the decryption stage, we only get g^m. So we need to ensure the message m to be small (poly-size) in order to get m using brute force logarithm.

Here is how we support addition on the modified version of ElGamal:

- Input: $\mathsf{ct}_1 = (g^{r_1}, u_1 = h^{r_1} \cdot g^{m_1})$ and $\mathsf{ct}_2 = (g^{r_2}, u_2 = h^{r_2} \cdot g^{m_2})$.
- Output: $\mathsf{ct} = (g^{r_1+r_2}, u = u_1 \cdot u_2 = h^{r_1+r_2} \cdot g^{m_1+m_2})$.

Notice if ct_1 and ct_2 are encrypted using the same public key, then we can use the corresponding secret key to decrypt ct. This scheme which supports addition is also known as Additively Homomorphic Encryption.

Notice that using addition with multiplication, we will be able to support any function we want to compute. However, we need a different approach to support multiplication.

Definition 11.1 (Composite-order pairing groups)**.** We call a symmetric pairing group $(\mathbb{G}, \mathbb{G}_T, e)$ with order N as a symmetric composite-order pairing group if $N = pq$ where p and q are two large primes.

Remark 11.2. Recall those important properties of composite-order pairing groups:

- N is public while p and q are secret.
- Let g be a generator of $\mathbb{G}$. Define $g_p = g^q$, then the order of g_p is p. And we can define $\mathbb{G}_p$ as the subgroup of order p generated by g_p.
- Similarly, we can define $g_q = g^p$ with order q as well as the subgroup $\mathbb{G}_q$.

Boneh-Goh-Nissim (BGN). We now introduce the Boneh-Goh-Nissim (BGN) scheme [BGN05], which supports multiple additions and one multiplication on the ciphertexts.

- $\mathsf{KeyGen}() \to \mathsf{pk}, \mathsf{sk}$: Sample $N = pq$ and a composite-order pairing groups $(\mathbb{G}, \mathbb{G}_T, e)$. Also Sample $x \xleftarrow{R} Z_N$ and set $\mathsf{pk} = (g, h = g_q^x)$, $\mathsf{sk} = q$.
- $\mathsf{Encrypt}(\mathsf{pk}, m) \to \mathsf{ct}$: Sample $r \xleftarrow{R} \mathbb{Z}_p$ and set $\mathsf{ct} = (g^r, u = h^r \cdot g^m)$.
- $\mathsf{Decrypt}(\mathsf{sk}, ct) \to m$: Compute $u^q = g^{mq}$ and find m such that $g_p^m = u^q$

Here is how we support addition, it is exactly the same as what we do for Modified ElGamal:

- Input: $\mathsf{ct}_1 = (g^{r_1}, u_1 = h^{r_1} \cdot g^{m_1})$ and $\mathsf{ct}_2 = (g^{r_2}, u_2 = h^{r_2} \cdot g^{m_2})$.
- Output: $\mathsf{ct} = (g^{r_1+r_2}, u = u_1 \cdot u_2 = h^{r_1+r_2} \cdot g^{m_1+m_2})$.

Here is how we support multiplication:

- Input: $\mathsf{ct}_1 = (g^{r_1}, u_1 = h^{r_1} \cdot g^{m_1})$ and $\mathsf{ct}_2 = (g^{r_2}, u_2 = h^{r_2} \cdot g^{m_2})$.
- Output: $\mathsf{ct} = (g^{r_1+r_2}, u)$ where $u = e(h^{r_1} g^{m_1}, h^{r_2} g^{m_2})$.

Notice that we need to pair two messages to output the ciphertext for multiplication, which means we are only allowed to do 1 multiplication.

Correctness. We actually need to prove correctness for the encryption scheme as well as addition and multiplication. While the correctness for the encryption scheme and the addition is trivial, we will only prove the correctness for multiplication.
Assume we encrypt ct_1 and ct_2 using the same public key, now with the corresponding secret key q.

$$\begin{aligned} u^q &= e(h^{r_1} g^{m_1}, h^{r_2} g^{m_2})^q \\ &= e(h^{r_1}, h^{r_2} g^{m_2})^q e(g^{m_1}, h^{r_2})^q e(g^{m_1}, g^{m_2})^q \\ &= e(g^{m_1}, g^{m_2})^q \\ &= e(g, g)^{m_1 m_2 q} \end{aligned}$$

The third equation is because $e(h^{r_1}, h^{r_2} g^{m_2})$ and (g^{m_1}, h^{r_2}) are elements in order q subgroup of $\mathbb{G}_T$ since $h = g_q$. Thus we can use brute force logarithm to find $m_1 \cdot m_2$ where $e(g, g_p)^{m_1 \cdot m_2} = u^q$.

Security. We will use the same context of semantic security as the one we used for ElGamal encryption scheme. And the proof of semantic security for BGN scheme will be very similar to the proof for semantic security for ElGamal, except we need to rely on a different security assumption.

Subgroup decision assumption. For a composite-order group $\mathbb{G}$ with order $N = pq$, the subgroup decision hardness is:

$$\left(g, g_q^s, g_q^r\right) \overset{c}{\approx} \left(g, g_q^s, g^r\right),$$

where $r, s \xleftarrow{R} \mathbb{Z}_p$. And we don't know either p or q or g_p.

Now, we are switching context to Integrity of Computations by introducing the concept of Non-interactive zero-knowledge (NIZK) proof systems. In this context, zero-knowledge proof stands for proving a statement x is true without revealing anything more about x.

Definition 11.3 (NP Languages)**.** We call $\mathbb{L}$ as an NP language if there exists an efficient relation R such that:

$$x \in \mathbb{L} \iff \exists w : R(x, w) = 1$$

Sketch of proof systems for NP languages.

- The prover takes x, w as input and produce a "proof" π.
- The verifier takes x as input and receives the "proof" π. It outputs $1/0$ depending on whether $R(x, w) = 1$ or not.

Definition 11.4 (NIZK proof for an NP language $\mathcal{L}$)**.** A NIZK proof for an NP language $\mathcal{L}$ is a tuple of three efficient algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$: On input the security parameter λ, the setup algorithm outputs the common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$: On input the common reference string crs, the statement x, and a witness w, the prove algorithm outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi) \to 1/0$: Output the verified result of proof π.

Requirements. As we have correctness and security requirements for encryption scheme, we will have two requirements for a proof system:

- Completeness: Suppose $R(x, w) = 1$. If $crs \leftarrow \mathsf{Setup}()$ and $\pi \leftarrow \mathsf{Prove}(crs, x, w)$, we have $\mathsf{Verify}(crs, x, \pi) = 1$.
- Soundness: For all adversary A (we don't assume efficiency here):

$$\Pr[x \notin \mathbb{L} \cap \mathsf{Verify}(crs, x, \pi) = 1 : crs \leftarrow \mathsf{Setup}(), \pi \leftarrow \mathsf{Prove}(crs, x, w)] \leq \mathsf{negl}(\lambda)$$

Above is only the requirements for a valid proof system, we still need a definition for zero-knowledge.

Definition 11.5 (Zero-Knowledge)**.** There exists an efficient simulator $S = (S_0, S_1)$ such that for all efficient A:

$$|w_0 - w_1| \leq \mathsf{negl}(\lambda)$$

where w_0 and w_1 are defined as follows:

- **Real distribution**
$$w_0 = \Pr[A^{O_o(crs,\cdot,\cdot)}(crs) = 1 : crs \leftarrow \mathsf{Setup}()]$$
- **Ideal distribution**
$$w_1 = \Pr[A^{O_1(st,\cdot,\cdot)}(crs) = 1 : (crs, st) \leftarrow S_0]$$

O_0 and O_1 are oracles defined as:

- $O_0(crs, x, w)$: If $R(x, w) = 1$: output $\mathsf{Prove}(crs, x, w)$, otherwise: output $\perp$
- $O_1(st, x, w)$: If $R(x, w) = 1$: output $S_1(st, x)$, otherwise: output $\perp$

Groth-Ostrovsky-Sahai (GOS) NIZK proof system sketch [GOS06]. The GOS NIZK proof system reduces the task of constructing a NIZK for a general NP language to constructing a NIZK for a related boolean circuit consisting of only NAND gates as follows.

1. Let C be a boolean circuit which computes an NP relation R, assuming without loss of generality that C consists only of NAND gates (which are universal).
2. Firstly, the prover encrypts (or rather commits to) the values of each wire in C.
3. Next, the prover provides the verifier with the randomness for the statement and output wires (allowing the verifier to observe that the output of the circuit was "1" and that the correct statement was used; both of these are public pieces of information).
4. Finally, the prover argues that each of the NAND gates was computed correctly, which is sometimes referred to as "internal consistency".

CS 395T: Topics in Cryptography February 22, 2024

Lecture 12: Non-Interactive Zero Knowledge

Lecturer: David Wu *Scribe: Zeki Gurbuz*

Recall at the end of the previous lecture that we covered the high-level ideas behind the Groth-Ostrovsky-Sahai (GOS) Non-interactive Zero Knowledge (NIZK) proof system, which reduces the task of constructing a NIZK for a general NP language to constructing a NIZK for a related boolean circuit consisting of only NAND gates as follows [GOS06]:

1. Let C be a boolean circuit which computes an NP relation R, assuming without loss of generality that C consists only of NAND gates (which are universal).
2. Firstly, the prover encrypts (or rather commits to) the values of each wire in C.
3. Next, the prover provides the verifier with the randomness for the statement and output wires (allowing the verifier to observe that the output of the circuit was "1" and that the correct statement was used; both of these are public pieces of information).
4. Finally, the prover argues that each of the NAND gates was computed correctly, which is sometimes referred to as "internal consistency".

In this lecture, we will introduce the "commit and prove" paradigm, give a more detailed description of GOS, which will make use of the Boneh-Goh-Nissim (BGN) encryption scheme and its homomorphic properties which we covered previously, and finally motivate the idea of batch arguments (BARGs), the topic of our next lecture.

The commit-and-prove paradigm. In terms of the boolean circuit C defined above, the commit and prove paradigm describes proof systems that follow the following general protocol:

1. The prover commits to a statement x and a witness w.
2. Next, the prover proves that the commited values satisfy the relation $C(x, w) = 1$.

While we haven't formally covered commitments in this course, a helpful intuitive notion are "sealed envelopes", which physically give the relevant guarantees that:

- Once a message is written down and put into the sealed envelope, the message can't be altered (that is, once the envelope is opened, the message observed will be the same message which was originally written down).
- Until the envelope is unsealed, its contents are hidden from the outside world.

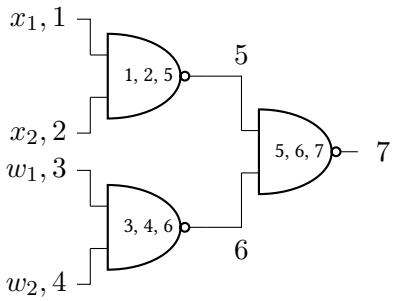
One example of the commit and prove paradigm is the prototypical *interactive* zero-knowledge proof system example of graph 3-coloring:

1. The prover will commit to the color of each node in the graph.

2. The verifier will ask to observe the colors of two nodes incident to a particular edge.

If the graph is indeed 3-colorable, then the verifier will always observe consistency. Otherwise, at least one edge in the graph would reveal a contradiction. As such, repeating the above protocol many times (while the prover repeatedly permutes the three colors it uses as to maintain zero-knowledge) can give a verifier which chooses a random edge each time strong confidence that the graph is indeed 3-colorable.

The Groth-Ostrovsky-Sahai (GOS) construction. We now turn our attention back to the GOS construction. Throughout the following description, $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ will represent a boolean circuit, consisting solely of NAND gates, which computes an NP relation R. C takes as input an n-bit statement as well as an h-bit witness and produces a single bit corresponding to $R(x,w)$. As an example, C might look like so (where here $n = h = 2$):



Notice in particular that each wire has been given an index. Let s denote the total number of wires in C. The indices $1, 2, \ldots, n$ will then index the statement, the indices $n+1, n+2, \ldots, n+h$ will index the witness, and finally the indices $n+h+1, n+h+2, \ldots, s$ will index the non-input wires. Furthermore, we will assume that all of the wires are indexed in a topological ordering. In particular, the prover can always compute the value of wire i by knowing the values of the wires $1, 2, \ldots, i-1$ and the structure of the circuit. Clearly, this forces wire s to be the output wire. Notice that each NAND gate is also labeled with a triple of wire indices, the first two being the input wires and the last being the output wire.

Next, the prover will compute $C(x,w)$. We denote $t_1, t_2, \ldots, t_s \in \{0,1\}$ to be the values of the wires during the computation. The prover now commits to the values $t_1, t_2, \ldots, t_n$. To do so, it will use the BGN encryption scheme, that is, for each $i \in [s]$, the prover samples $r_i \xleftarrow{\text{R}} \mathbb{Z}_N$ and sets $c_i \leftarrow g^{t_i} \cdot h^{r_i}$. The following tasks remain for the prover:

- It must show that c_s commits to the value 1 (proving that $C(x,w) = 1$).
- It must show that $c_1, c_2, \ldots, c_n$ commits to the statement x.
- It must show "internal consistency", that is:
  - It must show that for all $i \in [s]$, c_i commits to either 0 or 1.
  - It must show that each NAND gate is computed correctly.

The first two requirements are simple; the fact that $C(x,w)$ should evaluate to 1 and the statement itself are public information, so the prover can include $r_1, r_2, \ldots, r_n$ and r_s in the common reference string. From there, the verifier can itself check for consistency. The challenge is to show internal consistency.

Commitments to 0/1. The main remaining task is to show that for all $i \in [s]$, c_i commits to either 0 or 1. Let us first consider a fixed index $i \in [s]$. Then our BGN commitment is of the form $c = g^t \cdot h^r$. If $t \in \{0, 1\}$, then either $c = h^r$ or $c \cdot g^{-1} = h^r$. We now introduce a composite-order pairing group $(\mathbb{G}, \mathbb{G}_T, e)$ where $|G| = N = pq$ where p, q are large primes. By the "projection" property of composite-order pairing groups, then for an honest prover, $e(c, c \cdot g^{-1})$ lives in an order-q subgroup of $\mathbb{G}_T$, since one of c and $c \cdot g^{-1}$ live in $\mathbb{G}_q$. This implies that $e(c, c \cdot g^{-1}) = e(h, u)$ for some $u \in \mathbb{G}$, since $\langle h \rangle = \mathbb{G}_q$.

In particular, we can explicitly calculate this value $u \in \mathbb{G}$ for when $t = 0$ and when $t = 1$:

- If $t = 0$, then $e(c, c \cdot g^{-1}) = e(h^r, h^r \cdot g^{-1}) = e(h, (h^r \cdot g^{-1})^r)$, implying that $u = (h^r \cdot g^{-1})^r$.
- If $t = 1$, then $e(c, c \cdot g^{-1}) = e(g \cdot h^r, h^r) = e(h, (h^r \cdot g)^r)$, so $u = (h^r \cdot g)^r$.

More generally, $u = (h^r \cdot g^{2t-1})^r$ for $t \in \{0, 1\}$. Our initial approach, which will have a specific flaw, will be for the prover to provide u_i for each $i \in [s]$, and then have the verifier check that $e(c_i, c_i \cdot g^{-1}) = e(h, u_i)$ for each $i \in [s]$. To see why this is sound, we will argue that if $c = g^t \cdot h^r$ for some $t \notin \{0, 1\}$, then such a u *does not even exist.* As a slight aside, note that expressing c in this form binds the value of $t \bmod p$; it is not possible to have that $c = g^t \cdot h^r = g^{t'} \cdot h^{r'}$ since h is in the order-q subgroup. Then, we compute that

$$e(c, c \cdot g^{-1}) = e(g^t \cdot h^r, g^{t-1} \cdot h^r) = e(g, g)^{t \cdot (t-1)} \cdot e(g^t, h^r) \cdot e(g^{t-1}, h^r) \cdot e(h^r, h^r).$$

Notice that the latter three terms in this expression all live in the order-q subgroup of $\mathbb{G}_T$, whereas $e(g, g)^{t \cdot (t-1)}$ lives in the full group $\mathbb{G}_T$. In particular, if $t \notin \{0, 1\}$, then $t \cdot (t - 1) \neq 0$, so $e(c, c \cdot g^{-1})$ is non-identity in the order-p subgroup. On the other hand, $e(h, u)$ would live in the order-q subgroup, so the verification relation can never be satisfied.

As mentioned, this approach does have a flaw – it is not zero knowledge! This is because of the fact that there is no randomness. The intuition for why this is is similar to the idea that no deterministic encryption scheme can be semantically secure. We need to randomize somehow, and to do so we will pair with h^α (where α is chosen randomly) instead of pairing with h. Suppose we pick $\alpha \xleftarrow{R} \mathbb{Z}_N^*$. Then $e(h, u) = e(h^\alpha, u^{\alpha^{-1} \bmod N})$, and we can replace our verification relation accordingly.

We will begin to build our proof as $\pi_1 \leftarrow h^\alpha$ and $\pi_2 \leftarrow h^{\alpha^{-1}}$, and use the verification relation $e(c, c \cdot g^{-1}) \stackrel{?}{=} e(\pi_1, \pi_2) = e(h, u)$. But now we have introduced another problem – this change to our verification relation breaks soundness! The adversary could simply pick π_1 to be an element of the order-p subgroup, trivializing the verification relation. As such, we will also include in our proof $\pi_3 \leftarrow g^\alpha$. Then, we additionally check $e(\pi_1, g) \stackrel{?}{=} e(\pi_3, h)$. Since g and π_3 do not live in the order-q subgroup and h does, this forces π_1 to live in the order-q subgroup as desired.

The informal argument for why this randomized version of our original idea is indeed zero knowledge is that we can "lift" h to the full group using the assumed hardness of the subgroup decision problem. Then, it becomes true that π_1 fully determines the values of π_2 and π_3.

Commitments to NAND consistency. The other remaining task to show internal consistency is to argue that each NAND gate is computed correctly. We can describe each NAND gate as a tuple of three elements (i, j, k) where $i, j, k \in [s]$ and $i < j < k$. The first two values of the tuple represent the indices of the input wires to the NAND gate, and the third value represents the index of the output wire. As such, our goal will be to show that $t_k = \mathsf{NAND}(t_i, t_j)$.

The convenient position we find ourselves in is that we have already shown that $t_i, t_j, t_k \in \{0, 1\}$. In particular, for $t_k = \mathsf{NAND}(t_i, t_j)$ if and only if $t_k = 1 - t_i \cdot t_j$, which for $t_i, t_j, t_k \in \{0, 1\}$, is if and only if $t_i + t_j - 2 \cdot t_k + 2 \in \{0, 1\}$. This can be seen via case analysis, but we do not include a proof.

Our strategy will be to derive a commitment to $t_i + t_j - 2 \cdot t_k + 2$ using the linear homomorphism of BGN encryption, and then use our same strategy to argue that this commitment is to either zero or one. We compute $c_i \cdot c_j \cdot c_k^{-2} \cdot g^2 = g^{t_i + t_j - 2 \cdot t_k + 2} \cdot h^{r_i + r_j - 2 \cdot r_k}$, the desired commitment. In particular, we *do not* prepare this commitment using multiplicative homomorphism, since this would leave us stuck in the target group with no hope of using our previous strategy.

Finally, we will give a formal description of GOS:

Construction 12.1 (The Groth-Ostrovsky-Sahai (GOS) NIZK Proof System)**.**

- Setup(): The common reference string is prepared as $\mathsf{crs} \leftarrow ((\mathbb{G}, \mathbb{G}_T, e), N = pq, g, h \in \mathbb{G}_q, C)$ for a composite-order pairing group of order $N = pq$ where p, q are large primes, $\langle g \rangle = \mathbb{G}$, $\langle h \rangle = \mathbb{G}_q$, and $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ a circuit consisting of solely NAND gates which computes the NP relation $R(x, w)$.
- Prove($\mathsf{crs}, x \in \{0,1\}^n, w \in \{0,1\}^h$) :
  1. Let $t_1, t_2, \ldots, t_s$ be the wires of $C(x, w)$ in topological ordering.
  2. For each $i \in [s]$, sample $r_i \xleftarrow{R} \mathbb{Z}_N$ and let $c_i \leftarrow g^{t_i} \cdot h^{r_i}$.
  3. For each $i \in [s]$,
     (a) Sample $\alpha_i \xleftarrow{R} \mathbb{Z}_N^*$.
     (b) Let $\pi_i^{(1)} \leftarrow h^{\alpha_i}$, $\pi_i^{(2)} \leftarrow h^{{\alpha_i}^{-1}}$, and $\pi_i^{(3)} \leftarrow g^{\alpha_i}$.
     (c) Let $\pi_i \leftarrow \left(\pi_i^{(1)}, \pi_i^{(2)}, \pi_i^{(3)}\right)$.
  4. For each $(i, j, k) \in C$,
     (a) Sample $\alpha_{(i,j,k)} \xleftarrow{R} \mathbb{Z}_N^*$.
     (b) Let $c_{(i,j,k)} \leftarrow c_i \cdot c_j \cdot c_k^{-2} \cdot g^2$.
     (c) Let Let $\pi_i^{(1)} \leftarrow h^{\alpha_{(i,j,k)}}$, $\pi_i^{(2)} \leftarrow h^{{\alpha_{(i,j,k)}}^{-1}}$, and $\pi_i^{(3)} \leftarrow g^{\alpha_{(i,j,k)}}$.
     (d) Let $\pi_{(i,j,k)} \leftarrow \left(\pi_{(i,j,k)}^{(1)}, \pi_{(i,j,k)}^{(2)}, \pi_{(i,j,k)}^{(3)}\right)$.
  5. Output $\pi \leftarrow (c_1, c_2, \ldots, c_s, \{c_{(i,j,k)}\}_{(i,j,k) \in C}, r_1, r_2, \ldots, r_n, r_s, \{\pi_i\}_{i \in [s]}, \{\pi_{(i,j,k)}\}_{(i,j,k) \in C})$.
- Verify($\mathsf{crs}, x \in \{0,1\}^n, \pi$) :
  1. Parse π as $(c_1, c_2, \ldots, c_s, \{c_{(i,j,k)}\}_{(i,j,k) \in C}, r_1, r_2, \ldots, r_n, r_s, \{\pi_i\}_{i \in [s]}, \{\pi_{(i,j,k)}\}_{(i,j,k) \in C})$.
  2. Let $t_i \leftarrow \begin{cases} 0 & c_i = h^{r_i} \\ 1 & c_i = g \cdot h^{r_i} \text{ for each } i \in [n] \cup \{s\}. \\ \perp & \text{otherwise} \end{cases}$
  3. If $\perp \in \{t_1, t_2, \ldots, t_n, t_s\}$, or $t_1 || t_2 || \ldots || t_n \neq x$, or $t_s = 0$, output 0.
  4. For each $i \in [s]$, if $e(c_i, c_i \cdot g^{-1}) \neq e\left(\pi_i^{(1)}, \pi_i^{(2)}\right)$ or $e\left(\pi_i^{(1)}, g\right) \neq e\left(\pi_i^{(3)}, h\right)$, output 0.
  5. For each $(i, j, k) \in C$, $e(c_{(i,j,k)}, c_{(i,j,k)} \cdot g^{-1}) \neq e\left(\pi_{(i,j,k)}^{(1)}, \pi_{(i,j,k)}^{(2)}\right)$ or $e\left(\pi_{(i,j,k)}^{(1)}, g\right) \neq e\left(\pi_{(i,j,k)}^{(3)}, h\right)$, output 0.
  6. Otherwise, as no verification relations have failed, output 1.

Batch arguments (BARGs). Finally, we introduce the concept of batch arguments (BARGs). In this setting, we have T many statements $x_1, x_2, \ldots, x_T$, and the goal is to prove that there exists witnesses $w_1, w_2, \ldots, w_T$ such that $C(x_i, w_i) = 1 \; \forall \; i \in [T]$. The interesting quality that we ask for is "succintness", that $|\pi|$ is independent on the value of T. In the next lecture, we will show how to achieve $|\pi| = \mathsf{poly}(\lambda, |C|)$.

CS 395T: Topics in Cryptography February 27, 2024

Lecture 13: Batch Arguments

Lecturer: David Wu *Scribe: Alex Huang*

Up until this point in the course, we have focused on using cryptography to achieve privacy. For the next few lectures, we will instead focus on using cryptography to achieve efficiency/succinctness.

Motivation: Verifiable Computation In the case where a the computation of program P on some value x would be computationally expensive, we would like to ask a more powerful server to perform the computation. We can model this as a client with the program and value as parameters $C(P, x)$. However, we want to verify that the server's computation is legitimate so we can trust the output of the server. Therefore, the verifiable computation procedure can be described as:

1. Client sends (P, x) to the server.
2. Server responds with $(y = P(x), \pi)$, where π is a proof that the computation is correct.

Verifying the proof A trivial way to verify the computation would be to compute $P(x)$ locally. However, this defeats the purpose of using the server for the computation. Therefore, we would like to enforce the property that verifying the proof π is a much easier computation than computing $P(x)$.

Requirements We will have three requirements for our computation verification system:

1. Efficiency/Succinctness: Checking the proof must be much easier than computing $P(x)$. Formally, we require that $T_{\mathsf{Verify}} = \mathsf{poly}(\lambda, \log T_P)$, where T_P is the running time of program P.
2. Completeness: If $y = P(x)$, then the verifier should accept the proof π on value y with respect to (P, x).
3. Soundness: If $y \neq P(x)$, then the verifier should reject the proof π with probability $1 - \mathsf{negl}$.

It turns out that verifiable computation requires an intermediate notion to first be achieved, so we will delay describing the full construction until the next lecture. In fact, verifiable computation was not realized until 2021, but it makes use of previous primitives, one of which is the concept of a batch argument.

Batch Arguments for NP (BARG) BARG is an aggregation system for proofs, analogous to how BLS signatures aggregate signatures or BGW broadcast encryption aggregates ciphertexts. The setting for a BARG is a collection of T statements in an NP language $x_1, \ldots, x_T$, and a circuit C. We would like to prove that for all $i \in [T]$, there exists a witness w_i such that $C(x_i, w_i) = 1$. This can be accomplished trivially by sending $(w_1, w_2, \ldots, w_T)$, just like in a NIZK, but it renders the construction useless. Unlike in the zero-knowledge construction, the requirement for a BARG is succinctness.

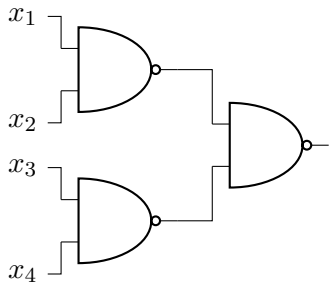
Succintness for a BARG The proof π for a BARG is succinct if $|\pi| = \mathsf{poly}(\lambda, C)$. This notion of succinctness is hiding and actually implies NIZK. Informally, this is because since the proof has fewer than T bits, it is necessarily a compression which loses information and therefore cannot contain knowledge of the witnesses. This implication is a relatively recent discovery.

Remark 13.1. It is also possible to achieve $|\pi| = \mathsf{poly}(\lambda)$, where the proof is independent of the circuit size. However, this construction is much harder to achieve and beyond the scope of this lecture, requiring more advanced cryptographic tools.

Construction 13.2 (Waters-Wu BARG [WW22])**.** The construction of our BARG system will be based off the same principles as the GOS proof system from the previous lecture. Namely, we will employ a commit-and-prove system. Unlike GOS, though, we will need the committment scheme to be compressing. Since the GOS commitment depends on T, we cannot use it, as it will scale with the size.

Ingredients for the BARG construction

1. **A succinct way to commit:** In our setting where we have T copies or instances of a circuit, we would like to compress our proof to be independent of T. For a circuit such as the following, we want to commit the vectors $(x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(T)})$ for $i \in [4]$.



   In the GOS construction, to aggregate proofs we committed to the value $g^{x_i} h^r$. Here, we can use multiplication to instead commit to a vector. We define the common reference string crs to include the generator g, as well as T random group elements $h_1, h_2, \ldots, h_T \xleftarrow{R} \mathbb{G}$ (where h_i denotes g^{α_i} for some random α_i), instead of just one random group element in the GOS construction. To commit to a vector $v = (v_1, v_2, \ldots, v_T)$, we define the commitment $c = \prod_{i \in [T]} h_i^{v_i}$, giving us a single group element whose size is independent of T.

2. **Committing across instances:** For each wire j in the circuit C, we let $(t_j^{(1)}, t_j^{(2)}, \ldots, t_j^{(T)})$ be the values across the T instances. Then, we can commit to $c_j \leftarrow \prod_{i \in [T]} h_i^{t_j^{(i)}}$ for all j.

Together, these ingredients inform us of the structure of the common reference string and an individual commitment, where $\mathsf{crs} = (g, h_1, h_2, \ldots, h_T)$ and a $c_j = \prod_{i \in [T]} h_i^{t_j^{(i)}}$ for all $j \in [S]$.

Internal consistency properties Similarly to the GOS construction, we want to be able to show the following two properties for parts of the circuit:

1. **Completeness:** For all $i \in [T]$ and $j \in [S]$, $t_j^{(i)} \in \{0,1\}$.

   Suppose $c = \prod_{i\in[T]} h_i^{v_i}$; then, we want to show that $v_i \in \{0,1\}$ for all $i \in [T]$. As before in the GOS construction, we can use the property that $v_i \in \{0,1\} \leftrightarrow v_i \cdot (v_i - 1) = 0 \leftrightarrow v_i^2 = v_i$. Then we have

$$\begin{aligned} e(c,c) &= e(\prod_{i\in[T]} h_i^{v_i}, \prod_{i\in[T]} h_i^{v_i}) \\ &= e(g^{\sum_{i\in[T]} \alpha_i v_i}, g^{\sum_{i\in[T]} \alpha_i v_i}) \\ &= e(g,g)^{(\sum_{i\in[T]} \alpha_i v_i)^2} \\ &= e(g,g)^{(\sum_{i\in[T]} \alpha_i^2 v_i^2) + (\sum_{i\in[T]} \sum_{j\neq i} \alpha_i \alpha_j v_i v_j)} \end{aligned}$$

   If we have that $v_i^2 = v_i$, then $\sum_{i\in[T]} \alpha_i^2 v_i^2 = \sum_{i\in[T]} \alpha_i^2 v_i$. We can compute this in the exponent using pairings based on the following derivation:

$$\begin{aligned} e(c, \prod_{i\in[T]} h_i) &= e(g^{\sum_{i\in[T]} \alpha_i v_i}, g^{\sum_{i\in[T]} \alpha_i}) \\ &= e(g,g)^{(\sum_{i\in[T]} \alpha_i v_i)(\sum_{i\in[T]} \alpha_i)} \\ &= e(g,g)^{(\sum_{i\in[T]} \alpha_i^2 v_i) + (\sum_{i\in[T]} \sum_{j\neq i} \alpha_i \alpha_j v_i)} \end{aligned}$$

   Taking these two facts together, we can now produce the verification relation

$$e(c,c) \stackrel{?}{=} e(c, \prod_{i\in[T]} h_i) \cdot e(g,g)^{\sum_{i\in[T]} \sum_{j\neq i} \alpha_i \alpha_j (v_i v_j - v_i)}$$

   The first two pairings can be computed by the verifier, but the last pairing requires knowledge of certain exponents which we do not know a priori. Therefore, we can add the group elements $u_{ij} = g^{\alpha_i \alpha_j}$ for all $i \neq j$ to the common reference string. Then, $e(g,g)^{\sum_{i\in[T]} \sum_{j\neq i} \alpha_i \alpha_j (v_i v_j - v_i)} = e(g, \prod_{i\in[T]} \prod_{j\neq i} u_{ij}^{v_i v_j - v_i})$, which is now a pairing computable by the verifier with the additional common reference string elements.

2. **Gate consistency:** All NAND gates should be correctly computed.

   Take any three wires $v_{1,i}, v_{2,i}, v_{3,i}$. For this check, if we assume we passed the first check, then $v_{j,i} \in \{0,1\}$ for $j \in \{1,2,3\}$. Therefore, an equivalent formulation for $v_{3,i} \stackrel{?}{=} v_{1,i} \tilde{\wedge} v_{2,i}$ is to check that $v_{1,i} + v_{2,i} - 2v_{3,i} + 2 \in \{0,1\}$. To do this, note that we have access to the commitments $c_1 = \prod_{i\in[T]} h_i^{v_{1,i}}$, $c_2 = \prod_{i\in[T]} h_i^{v_{2,i}}$, and $c_3 = \prod_{i\in[T]} h_i^{v_{3,i}}$. By simply multiplying these together, we get

$$\begin{aligned} c^* &= c_1 \cdot c_2 \cdot c_3^{-2} \cdot \prod_{i\in[T]} h_i^2 \\ &= (\prod_{i\in[T]} h_i^{v_{1,i}}) \cdot (\prod_{i\in[T]} h_i^{v_{2,i}}) \cdot (\prod_{i\in[T]} h_i^{-2v_{1,i}}) \cdot (\prod_{i\in[T]} h_i^2) \\ &= \prod_{i\in[T]} h_i^{v_{1,i} + v_{2,i} - 2v_{3,i} + 2} \end{aligned}$$

   This allows us to verify the relation in the exponent. Note that this derivation also implies that c^* is a commitment to $v_{1,i} + v_{2,i} - 2v_{3,i} + 2$ for all $i \in [T]$. We can now check that this value is 0 or 1 using the technique described above.

Soundness How do we argue soundness? For this proof, we only know how to achieve a weaker notion of soundness, **non-adaptive soundness**, which is analogous to selective security. For non-adaptive security, the adversary commits to a false statement $(x_1, \ldots, x_T)$ before seeing the common reference string. Our approach for this proof will be to program a secret index $i^* \in [T]$ into the common reference string. Then, when given a valid proof for $(x_1, \ldots, x_T)$, we can extract a witness w_{i^*} for the x_{i^*} such that $C(x_{i^*}, w_{i^*}) = 1$, which suffices for non-adaptive soundness as long as i^* is hidden from the adversary, since it implies that the adversary's behavior will not change based on the choiec of i^*. This property of being statistically sound on one instance is called somewhere-soundness.

Remark 13.3. It is actually possible to use this to also prove adaptive soundness through a technique known as complexity leveraging, when complexity leveraging for index hiding.

Programming the CRS The normal CRS for this construction has $(g, h_i = g^{\alpha_i}, u_{ij} = g^{\alpha_i \alpha_j})$ for all $i \in [T]$ and $j \neq i$. For the proof, we want to view all of these group elements as elements in a composite-order group $(g, h_i = g_p^{\alpha_i}, u_{ij} = g_p^{\alpha_i \alpha_j})$ for all $i \in [T]$ and $j \neq i$. To bind the CRS, we replace h_{i^*} with $g^{\alpha_{i^*}}$, $u_{i^* j}$ with $h_{i^*}^{\alpha_j}$ and u_{iu^*} with $h_i^{\alpha_{i^*}}$, essentially "lifting" the index of interest to the full group. Using the subgroup decision problem, we can argue that this binding CRS is indistinguishable from the normal CRS. Now, we can extract from every commitment. When we bind the CRS, a commitment is now $c = \prod_{i \in [T]} h_i^{v_i} = h_{i^*}^{v_{i^*}} \cdot \prod_{i \neq i^*} h_i^{v_i}$. Note that $h_{i^*}^{v_{i^*}}$ is in the full group, whole $\prod_{i \neq i^*} h_i^{v_i}$ is in the order p subgroup. Then, the pairing $e(c, g_q) = e(h_{i^*}^{v_{i^*}}, g_q)$ holds, so we can solve discrete log to retrieve v_{i^*}.

Remark 13.4. The construction only ever uses elements from the order p subgroup. However, even though its existence is never used, we require the order q subgroup to exist for the security proof.

The full construction [WW22]

- To generate crs: Sample $\alpha_i \xleftarrow{R} \mathbb{Z}_p$ for all $i \in [T]$. Set $\mathsf{crs} \leftarrow (g, \{h_i\}_{i \in [T]}, \{u_{ij}\}_{i \neq j})$, where $h_i = g^{\alpha_i}$ and $u_{ij} = g^{\alpha_i \alpha_j}$.
- To commit to a vector $v = (v_1, \ldots, v_T)$: Output $c \leftarrow \prod_{i \in [T]} h_i^{v_i}$.
- To prove $v_i \in \{0, 1\}$: Output $\pi \leftarrow \prod_{i \in [T]} \prod_{j \neq i} u_{ij}^{v_i v_j - u_i}$.
- To check a proof π, verify that $e(c, c) \stackrel{?}{=} e(c, \prod_{i \in [T]} h_i) \cdot e(g, \prod_{i \in [T]} \prod_{j \neq i} u_{ij}^{v_i v_j - v_i})$.

CS 395T: Topics in Cryptography February 29, 2024

Lecture 14: RAM Delegation

Lecturer: David Wu *Scribe: Nate Mikosh*

So far, we have focused on proving properties in a privacy-preserving manner. Next, we will look at achieving short proofs that are efficient to verify.

Verifiable Computation In verifiable computation, the client provides a program P, and input x to some server. The server responds with a proof $y = P(x)$. The client needs to be able to check that this proof is correct, but checking this correctness should be much faster than if the client were to compute $P(x)$ itself. This process of using a server to run a program that the client can check for correctness is called delegation, or a succinct non-interactive argument (SNARG).

Our goal is to check that $y = P(x)$ was evaluated correctly. A reasonable approach is to break the program P up into individual instructions and use a batch argument to check that each instruction was executed properly. Before we can do this though, we should define P's structure. We will assume that P is a RAM program.

RAM Programs A RAM (random access machine) program is defined as follows:

- The program has access to some vary large M bits of memory and a small number of bits of local state st
- Memory is initialized as the input to the program x, with all other bits as 0s
- The program consists of a sequence of only reads and write, defined as follows:
  - Read a single bit of memory and update state
  - Write a single bit of memory and update state
- The program outputs the contents of memory at the end of computation

Now that P's structure is clear, we can break it up in order to use a batch argument. A simple way to do this is to separate the program into sequence of state and memory values. (st_0, M_0) represent the initial values of state and memory, (st_1, M_1) are the values after one instruction has been executed, and (st_T, M_T) are the final values where T is the length of the program. To check that each instruction in the program was executed properly, we can check that each consecutive pair of state and memory makes sense. We can now construct our batch argument as follows:

- IsValid: on inputs $((st_i, M_i), (st_{i+1}, M_{i+1}))$, output 1 if the transition is valid according to some criteria

  The exact criteria will vary, but for example:

  - For a read, checking that memory is unchanged
  - For a write, checking that at most one bit of memory was changed
- Statements: $((st_0, M_0), (st_1, M_1)), ...((st_{T-1}, M_{T-1}), (st_T, M_T))$

- Proof: BARG proof that IsValid holds for all statements

While this initially seems like an effective argument, there is no succinctness here. The goal of a SNARG is to verify correctness of a program much faster than actually running it, and this argument fails to achieve that. This issue can partially be solved by compressing our statements. While hashing would normally obscure the information we need for our argument, we will introduce a new method of hashing, Merkle hashing, to solve this problem.

Merkle Hashing Merkle hashing is a hashing technique that hashes several values in such a way that an "opening" can be used to reveal the value of an input without revealing any other inputs. To demonstrate this let's look at a Merkle hash on the inputs (x_1, x_2, x_3, x_4) where $H(x)$ is some collision-resistant hash function.
The inputs to a Merkle hash can be thought of as leaves of a binary tree, as we step through the hash, the tree will be built.

$$x_1, x_2, x_3, x_4$$

Each step of the Merkle hash generates another layer of the tree. A new "node" is generated for every pair of inputs.

$$h_{12} = H(x_1||x_2), h_{34} = H(x_3||x_4)$$

This process continues until we arrive at one final hash.

$$h_{14} = H(h_{12}||h_{34})$$

The final value of the hash, in this case h_{14}, is referred to as the root of the Merkle hash.
The utility of a Merkle hash is that we can we can create short local openings to prove the value of an input $y = x_i$ with respect to the root h. Once again think of the Merkle hash as a tree. By providing the necessary nodes, we re-create the root and verify that $y = x_i$. Using the same example as before, let's say we want to verify the value of x_1. The opening to x_1 would be the values of x_2 and h_{34}. We can then compute h_{12} and h_{14} ourselves and check that it matches the given root h. The size of these openings will only be $\lambda log(n) + l$ where λ is the length of the hash function output and l is the length of each input.

Improved BARG With this new hash function, we can construct a more succinct BARG. We can Merkle hash on $((st_0, M_0), ..., (st_T, M_T))$ to create a hash of all our statements "hstate". With hstate, we can construct a more succinct BARG as follows:

- IsValid: on $i \in [T]$
  - Witnesses: Merkle openings with repsect to hstate in order to get $(st_i, M_i), (st_{i+1}, M_{i+1})$
  - Checks that transition is valid according to criteria like in our previous version
- Statements: $i \in [T]$
- Proof: BARG proof that IsValid holds for all statements

The runtime of this argument is as follows:

- $|\pi BARG| = poly(\lambda, log(T), |IsValid|)$
- $|IsValid| = poly(\lambda, log(T), M)$

While this argument is slightly improved, the runtime of IsValid is based on the size of the program's memory. Memory for RAM programs is significantly large, therefore this argument still lacks succinctness. The solution is to use yet another Merkle hash, this time on each value of memory. Every state (st_i, M_i), can be translated into (st_i, h_i) by Merkle hashing M_i. We then hash every (st_i, h_i) pairing into hstate like we did in the previous version.

After this change, our final BARG proof is complete:

- IsValid: on $i \in [T]$
  - Witnesses: Merkle openings with respect to hstate in order to open $(st_i, h_i), (st_{i+1}, h_{i+1})$
  - To check a read, we check the Merkle opening with respect to h_i for bit j to check that it is correctly computed
  - For a write, we similarly check bit j, but then we can use this same opening to reconstruct h_{i+1} and check that it is also valid
- Statements: $i \in [T]$
- Proof: BARG proof that IsValid holds for all statements

This BARG is much faster. Opening each (st_i, h_i) with respect to hstate is $O(\lambda log(T))$, opening a bit with respect to some h_i is $O(\lambda log(M))$. Therefore, $|IsValid| = poly(\lambda log(T) + \lambda log(M))$ and $|\pi BARG|$ remains unchanged at $poly(\lambda, log(T), |IsValid|)$.

Security Merkle hashing on each memory state is acceptable, but a Merkle hash on all the states to generate hstate is actually not strong enough to guarantee security. For full security, we would need a somewhere extractable hash function which works similarly to a Merkle hash, but is stronger.

Extensions There are a couple interesting extensions of being able to produce SNARG for P, we'll look at two of them

- Homomorphic signature
  Given a signature on a message m, we can derive a signature on $f(m)$ for an arbitrary f. We sign m, the hash $h(m)$. The signature on $f(m)$ is the hash, the signature on the hash, the value $y = h(m)$ and a proof that that $y = h(m)$ with respect to $h(m)$.
- Functional Commitments
  With functional commitments, we can prove that a single function f is being used rather than using different functions for different values. We commit to an input x and then open to $f(x)$.

CS 395T: Topics in Cryptography March 5, 2024

Lecture 15: Polynomial Commitments

Lecturer: David Wu *Scribe: Noel Elias*

Previously, we showed how to construct verifiable computation/delegation - suitable for when the client knows the program P and input x. The client wants to learn P(x) and verify that this value was computed correctly in less time than it takes to compute P(x). This construction is a succinct non-interactive argument (SNARG) for a deterministic polynomial-time computation (SNARG for P).

In many cases however, the full input might not be known to the verifier. For example, suppose a server publishes a hash of some database. The client performs a query and wants a proof that the query was performed correctly relative to the hash of the database. Note the key difference in this setting is that the client does not know the entirety of the database, it only knows the hash. The trivial construction has no notion of soundness which binds the server's response to a database D.

So for this setting we need a SNARG for NP. Namely, we consider the following NP relation:

- Statement: Takes hash h of the database contents as well as y, the output of the query
- Witness: database (D)
- Relation: Checks that h=Hash(D) and y is output of query algorithm on D. Outputs 1 if all these properties hold.
- In particular, we want a proof size $|\pi| = poly(\lambda)$.

In this lecture, we will construct a SNARG for NP where the size of the proof is a constant number of group elements, regardless of the complexity of the NP relation. As a result, this construction will rely more assumptions including the random oracle (RO) as well generic group model. Recall, SNARGs rely on two components: a functional commitment scheme + informational-theoretic component. In this construction we will use a polynomial commitment scheme and a polynomial IOP.

Polynomial Commitment Scheme We will work specifically over $\mathbb{F}_p$ (integers modulo p). A polynomial commitment scheme over $\mathbb{F}_p$ consists of four algorithms:

- Setup (d): Takes the (max) degree of the polynomial and outputs a common reference string (crs)
- Commit(crs, f) $\rightarrow$ c: Commits to the polynomial f of degree at most d.
- Eval(crs, c, f, x) $\rightarrow \pi$: Computes an opening π to the evaluation y = f(x).
- Verify(crs, c, x, y, π) $\rightarrow$ 0/1: : Checks whether opening is valid or not.

We can show some properties of a polynomial commitment scheme as follows:

- Succinctness: The size of both $|c|$ and $|\pi|$ are O(1) group elements, regardless of the degree of f.
- Correctness: Let f(X)=$f_0 + f_1 X + ... + f_d X^d$ be a polynomial and $x \in \mathbb{F}_p$ be a point. If crs $\leftarrow$Setup(d), $c \leftarrow$Commit(crs, f), $\pi \leftarrow$Eval(crs, c, x), then Verify(crs, $c, x, f(x), \pi$)=1.

- Binding: Given crs, difficult to come up with commitment c, a point x, and value/opening pairs $(y_0, \pi_0), (y_1, \pi_1)$ where if $y_0 \neq y_1$ and Verify(crs, c, x, y_0, π_0)=1=Verify(crs, c, x, y_1, π_1). Note that this definition allows us to open to 2d points where no degree d polynomial goes through all these points. So for applications, we often need a stronger *knowledge soundness* property =. This property states that if an efficient A can open a commitment c at k points, there exists an efficient extractor that outputs a polynomial f of degree at most d such that $f(x_i) = y_i$ (as we focus more on succinctness knowledge soundness will not be too important for understanding the construction).

Using this general framework for polynomial commitment scheme, we can define the Kate-Zaverucha-Goldberg (KZG) polynomial commitment scheme construction.

Construction 15.1. (Kate-Zaverucha-Goldberg (KZG) Polynomial Commitment Scheme [KZG10]):

- Setup (d): Sample $\alpha \xleftarrow{R} \mathbb{F}_p$ where p is the order of the group and the scheme supports polynomials over $\mathbb{F}_p$. Output crs $= (g, g^{\alpha}, g^{\alpha^2}, ..., g^{\alpha^d})$
- Commit(crs, f): Commitment is the element $g^{f(\alpha)}$
  - Suppose f(X)=$f_0 + f_1 X + ... + f_d X^d$
  - Let crs $= (g_0, g_1, ..., g_d)$ where $g_i = g^{\alpha^i}$.
  - The commitment c=$\prod_{i=0}^{d} g_i^{f_i} = \prod_{i=0}^{d} g^{f_i \alpha^i} = g^{\sum f_i \alpha^i} = g^{f(\alpha)}$
- Eval(crs, c, f, $x*$): Let y=f(x). The goal is to construct a proof π that y=f(x*) where f is the polynomial associated with c.
  - Define the polynomial $\hat{f}(X) = f(X) - y$. Observe that f(x*)=y if and only if $\hat{f}$(x*)=0, or equivalently, if x* is a root of f. This means there exists a polynomial q(X) such that $\hat{f}$(X)=(X-x*)q(X).
  - The opening will be a commitment to the polynomial q(X)=$\sum_{i=0}^{d-1} q_i X^i$

$$\pi = \prod_{i=0}^{d-1} g_i^{q_i} = \prod_{i=0}^{d-1} g^{q_i \alpha^i} = g^{\sum q_i \alpha^i} = g^{q(\alpha)}$$

- Verify(crs, $c, x*, y, \pi$): Verifier will essentially check that the polynomials $\hat{f}$ and (X-x*)q(X) are equal at X=α.
  - Recall that if these polynomials are equal at the random point α, they are equal. Suppose f is a polynomial of degree at most d and is not identically 0. Then, $Pr_{x \xleftarrow{R} \mathbb{F}_p}[f(x) = 0] \leq d/p$.
  - Normally, we have that $c = g^{f(\alpha)}$ and $\pi = g^{q(\alpha)}$.
  - From c=$g^{f(\alpha)}$, verifier computes $c \cdot g^{-y} = g^{f(\alpha)-y} = g^{\hat{f}(\alpha)}$.
  - From $\pi = g^{q(\alpha)}$, verifier computes $e(g_1 g^{-x*}, \pi) = e(g^{\alpha - x*}, g^{q(\alpha)}) = e(g, g)^{(\alpha - x*)q(\alpha)}$
  - Verification relation thus checks if $e(g, c \cdot g^{-y}) = e(g_1 g^{-x*}, \pi)$. =

In general, it is true that if two polynomials f,g of degree at most d such that f(x)=g(x) at a random point $x \xleftarrow{R} \mathbb{F}_g$, then with probability $1 - d/p$ we can say that f=g. This generalizes to the *Schwartz–Zippel lemma* for multi-variate polynomials.

KZG Binding We can show the binding property of the KZG scheme by relying on the d-strong Diffie-Hellman assumption (given $g, g^{\alpha}, g^{\alpha^2}, ..., g^{\alpha^d}$), it is hard to come up with $(c, g^{\frac{1}{\alpha+c}})$ for any $c \neq -\alpha$.

Now suppose an adversary produces a commitment $c = g^S$ and opens c to two different values y_1 and y_2 at x* with proof $\pi_1 = g^{t_1}$ and $\pi_2 = g^{t_2}$ (note that the reduction does not know s, t_1, t_2). Then by the verification relation; $e(g, c \cdot g^{-y_1}) = e(g^{\alpha - x*}, \pi_1)$ and $e(g, c \cdot g^{-y_2}) = e(g^{\alpha - x*}, \pi_2)$. In the exponent, this means that $s - y_1 = t_1(\alpha - x*)$ and $s - y_2 = t_2(\alpha - x*)$. This allows for the following:

$$\begin{aligned} t_1(\alpha - x*) + y_1 - y_2 &= t_2(\alpha - x*) \\ (t_1 - t_2)(\alpha - x*) &= y_2 - y_1 \\ \frac{t_1 - t_2}{y_2 - y_1} &= \frac{1}{\alpha - x*} \end{aligned}$$

Thus, $g^{\frac{1}{\alpha - x*}} = g^{\frac{t_1 - t_2}{y_2 - y_1}} = (\pi_1/\pi_2)^{\frac{1}{y_2 - y_1}}$. Note $y_1 \neq y_2$ since the adversary needs to open c two different ways. If the adversary outputs x*=α, then the reduction trivially breaks the assumption. We will develop protocols to prove additional properties on committed polynomials. To motivate this, we first sketch the ideas underlying the PLONK scheme.

Arithmetic Circuits For the PLONK construction, the computational model will be arithmetic circuits. Gates will be addition or multiplication and wires will be labeled by a field element. This can be used for boolean circuits where we can implement AND and OR gates using multiplication and addition. An illustration is shown in Figure 15.1:

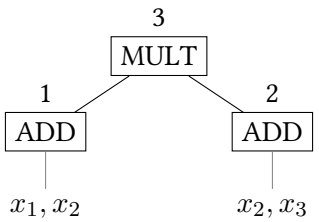


Figure 15.1

For any choice of input (x, x_2, x_3), we can define an *execution trace*. Every gate has a left input and a right input. Suppose $x_1 = 1, x_2 = 2, x_3 = 3$. Then the trace can be shown in Table 15.1:

Gate	Left Input	Right Input	Output
1	1	2	3
2	2	3	5
3	3	5	15

Table 15.1

PLONK The idea is for the PLONK system [GWC19] is that the prover will choose a polynomial that interpolates the entire execution trace.

More specifically, suppose we take a point $w \in \mathbb{F}_p$. Let m be a bound on the number of gates in the circuit and let n be the number of public inputs (i.e., the statement) that is known to the verifier. We require ord(w)>3m+n. Namely, the following elements are all distinct in $\mathbb{F}_p$: $w^{-n}, w^{-n+1}, ..., w^0, w^1, ..., w^{3m}$. The prover will interpolate the trace polynomial T where:

- $T(w^{-i})$ = value of ith public input
- $T(w^{3j})$ = value of left input to the jth gate
- $T(w^{3j+1})$ = value of right input to the jth gate
- $T(w^{3j+2})$ = value of output of jth gate

The polynomial T encodes the entire execution of C. The prover commits to C using a polynomial commitment scheme. The commitment is a single group element. Now the prover needs to show the following four checks to convince the verifier that the commitment of C is a valid trace for the computation:

1. Input Consistency: $T(w^{-i})$ = value of ith public input
2. Gate Consistency: Every gate is correctly implemented
3. Wire Consistency: If output of gate j is left input of gate k, then $T(w^{3j+2}) = T(w^{3k})$
4. Output Consistency: Output gate has the correct value.

In the next lecture we will dive into the details of implementing these checks for the PLONK construction.

CS 395T: Topics in Cryptography March 7, 2024

Lecture 16: Succinct Non-Interactive Arguments (SNARGs)

Lecturer: David Wu *Scribe: Noel Elias*

Recall in the previous lecture we described the KZG polynomial commitment scheme and gave an overview of the PLONK construction. In this lecture we will discuss the details of the PLONK construction [GWC19].

To review, we will examine the arithmetic circuit defined in the previous lecture in Figure 15.1. Our goal is to interpolate a polynomial that defines the execution trace of this arithmetic circuit as shown in Table 15.1.

To do this, we associate a field element with each value in the execution trace. Moreover, remember that n=length of statement while m=number of gates. The trace polynomial has the structure $w \in \mathbb{F}_p$ where ord(w)>3m+n. In addition, the following elements are all distinct in $\mathbb{F}_p$: $w^{-n}, w^{-n+1}, ..., w^0, w^1, ..., w^{3m}$. The prover will interpolate the trace polynomial T where:

- $T(w^{-i})$ = value of ith public input
- $T(w^{3j})$ = value of left input to the jth gate
- $T(w^{3j+1})$ = value of right input to the jth gate
- $T(w^{3j+2})$ = value of output of jth gate

To illustrate this polynomial structure we can label the execution trace with their equivalent polynomial as shown in Table 16.1:

Gate	Left Input	Right Input	Output
1	1 $(T(w^0))$	2 $(T(w^1))$	2 $(T(w^2))$
2	2 $(T(w^3))$	3 $(T(w^4))$	5 $(T(w^5))$
3	3 $(T(w^6))$	5 $(T(w^7))$	15 $(T(w^8))$

Table 16.1

Lastly, the prover must show that the commitment of this polynomial T satisfies the following checks to convince the verifier that this commitment of C is a valid trace for the computation. Ideally we want to do this with a constant number of group elements:

1. Input Consistency: $T(w^{-i})$ = value of ith public input
2. Gate Consistency: Every gate is correctly implemented
3. Wire Consistency: If output of gate j is left input of gate k, then $T(w^{3j+2}) = T(w^{3k})$
4. Output Consistency: Output gate has the correct value.

Output Consistency We can begin by showing how we check for output consistency in PLONK. Proving that the output gate has the correct value is just opening the polynomial commitment at $w^{3|c|-1}$.

Zero-Testing Gadget To prove consistency for the other three checks we will utilize a zero-testing gadget which shows that a polynomial f(X) is zero on a set S given only a polynomial commitment to f.

First, define the vanishing polynomial for S:$Z_S(X) = \prod_{i \in S}(X - i)$. Then f is zero on S if and only if there exists a polynomial q(X) such that $f(X) = Z_S(X) \cdot q(X)$. The prover has the polynomial f and can derive q(X) by dividing $f(X)/Z_S(X)$ (verifier knows $Z_S(X)$). Suppose the verifier only has a commitment to f. To prove that f is zero on S, we can use the following protocol:

1. Prover commits to the polynomial q of degree at most d-—S— where f(X)=$Z_S(X) \cdot q(X)$.
2. Verifier samples a random $r \xleftarrow{R} \mathbb{F}_p$
3. Prover opens commitments to f and q at r
4. Verifier checks that f(r) = $Z_S(r)q(r)$

To see why this is sound suppose f(X) is not zero on S. Then, there does not exist a polynomial g(X) such that f(X) = $Z_S(X) \cdot g(X)$. Consider the polynomial $h(X) = f(X) - Z_S(X) \cdot q(X)$. This is a polynomial of degree at most d and is not the zero polynomial. Thus, it has at most d roots. Then, $Pr_{r \leftarrow \mathbb{F}_p}[h(r) = 0] = d/p = negl$. Note that $h(r) = 0 \overset{f}{\leftrightarrow} (r) = Z_S(r)q(r)$. In other words, if f is not zero on all points S, then h(x) is a non-zero polynomial as well.

Also note that this proof consists of 3 group elements and 3 field elements. As long as the degree of the protocol is small relative to the size of the field, this protocol will be sound. This can be made non-interactive using the Fiat-Shamir Heuristic which hashes the inputs (commitments to f and q) to get the randomness r. So using the random oracle model we can make the proof non-interactive as well as only consist of three group elements: commitment to g, openings for f and q.

Input Consistency Now we can go back to the PLONK scheme where the prover has committed to a trace polynomial T. We can now analyze the input consistency.

- Suppose the public input is x=$(x_1, ..., x_n)$. Then, the prover should show that $T(w^{-i}) = x$ for all $i \in [n]$.
- To do so, prover (and verifier) interpolate polynomial v(X) where $v(w^{-i}) = x_i$.
- Then, the polynomial T(X)-v(X) is zero on the set S=$w^{-1}, ..., w^{-n}$. The prover and the verifier now run the zero-testing protocol described above.
- Note that in the zero-testing protocol, the prover needs to reveal T(r)=v(r). It does so by revealing T(r) and the verifer can then compute T(r)-v(r) itself.
- Thus, we can prove input consistency with a constant number of group elements.

Gate Consistency . For gate consistency we can define the following procedure:

- Define a selector polynomial v(X) where $v(w^{3j})$=1 if the gate j is an addition gate and $v(w^{3j})$=0 if the gate j is a multiplication gate.
- Suppose all the gates are implemented correctly. Then if gate j is an addition gate, $T(w^{3j+2}) = T(w^{3j}) + T(w^{3j+1})$. If gate j is a multiplication gate $T(w^{3j+2}) = T(w^{3j}) \cdot T(w^{3j+1})$.

- Combining both of these gates into one equation we get $T(w^{3j+2}) = [T(w^{3j}) + T(w^{3j+1})] \cdot v(w^{3j}) + [T(w^{3j}) \cdot T(w^{3j+1})] \cdot (1 - v(w^{3j}))$.
- So if gate constraints are satisfied, then for all $X \in [w^0, w^3, ..., w^{3|c|-3}]$, $T(w^2X)$ = v(S)[T(X)+T((wX)]+(1-v(S))[T(X)*T(wX)]).
- This reduces to the zero-test protocol on the set $[w^0, w^3, ..., w^{3|c|-3}]$.
- Note the implement this protocol, the verifier needs to evaluate polynomial $T(w^2X)$ = v(S)[T(X)+T((wX)]+(1-v(S))[T(X)*T(wX)]) at a random point r.
- In particular this can be implement using KZG by having the prover open T(r), T(wr), and T(w^2r) where the verifier can compute T(r) itself.
- Thus, gate consistency can be proven with a constant number of group/field elements as well.

Wire Consistency Recall the arithmetic circuit shown in Figure 15.1. We can rewrite the table trace with inputs $1(w^{-1})$, $2(w^{-2})$, $3(w^{-3})$ as shown in Table 16.2. We will show wire consistency as an example rather than a generic protocol:

Gate	Left Input	Right Input	Output
1	1 (w^0)	2 (w^1)	2 (w^2)
2	2 (w^3)	3 (w^4)	5 (w^5)
3	3 (w^6)	5 (w^7)	15 (w^8)

Table 16.2

In this example we require that $T(w^{-1}) = T(w^0)$, $T(w^{-2}) = T(w^1) = T(w^3)$, $T(w^{-3}) = T(w^4)$, $T(w^2) = T(w^6)$, and $T(w^5) = T(w^7)$. Whenever a wire value is used multiple times, we introduce a constraint. Every wire value participates in at most one constraint group. We can view this *replication pattern* as inducing a permutation P on the set $[w^{-3}, w^{-2}, .., w^8]$. For each input, w^i, $P(w^i)$ sends it to is where j is the index of the next copy of the wire associated with index i. P can be described by a polynomial of degree 3m+n (just like T). Checking equality of the wire constraints then boils down to checking T(X) = T(P(X)) for all $X \in [w^{-3}, w^{-2}, .., w^8]$. The polynomial P is known to the verifier so this can again be done using the zero-testing protocol.

However note that the this proof requires a quadratic sized circuit as well as a quadratic sized crs. To solve this in practice, we work with a bi-variate polynomial.

Summary To summarize to prove that C(x,w)=1, the prover commits to the execution trace T(X) of C and then proves the following checks: input consistency, gate consistency, wire consistency, and output consistency. Each proof requires revealing a constant number of group elements (i.e., commitments and openings to the polynomial commitment scheme). Soundness requires the random oracle (to make the interactive protocol non-interactive) and the algebraic group model (or generic group model) to argue soundness of the KZG scheme. Additionally, the crs of PLONK does not depend on the circuit and is thus known as having a *universal setup*. Properties like zero-knowledge can be easily added to the PLONK construction by making the KZG polynomial commitments implement hiding.

There are many extensions to the PLONK protocol. First, one can modify the base protocol so the prover complexity is quasi-linear in $|C|$ rather than quadratic. Second, one can consider working with multi-linear polynomials over the vector space $\mathbb{F}_2$ to support linear-time provers (HyperPlonk [CBBZ22]). Lastly, one can support more general gates by extending gate consistency checks with tools like look-up arguments.

CS 395T: Topics in Cryptography March 19, 2024

Lecture 17: Introduction to Lattices

Lecturer: David Wu *Scribe: Zeki Gurbuz*

Pairings Recap. Until now, we studied pairing-based cryptography, which gave us algebraic properties that led to numerous useful constructions such as the following:

- Short (BLS) Signatures,
- Identity-based Encryption (IBE),
- Broadcast Encryption,
- Attribute-based Encryption,
- Somewhat-homomorphic Encryption,
- Non-interactive Zero-knowledge (NIZK),
- and Succinct Non-interactive Arguments (SNARG),

many of which we did not know how to realize without pairings. At this point in the course, we'll pivot to *lattice-based cryptography*.

Lattices. There are many differences and trade-offs between lattice-based and pairing-based cryptography, and yet there are many subtle similarities as well. We'll see that lattices allow us to construct many of the same notions:

- IBE,
- ABE for circuits (hence arbitrary policies),
- Fully-homomorphic Encryption (FHE),
- NIZK,
- SNARG (which is a recent development),
- and more.

On the other hand, it is quite challenging to build, for example, broadcast encryption from standard lattice-based cryptographic assumptions! Only recent developments from new lattice-based assumptions have yielded broadcast encryption constructions. We also don't know how to build short signatures from lattices (it may not even be a well-defined notion), and these are major open topics of research.

Lattices have different algebraic properties/structure than pairings. A lot of pairing-based proofs required group-theoretic arguments, whereas lattices will revolve more around linear algebra. Moreover, our lattice calculations will generally be more in-the-weeds, requiring more notation, whereas pairing-based constructions did not have to manipulate e.g. elliptic curve operations, treating the pairing map as a black box.

Why Lattices? Lattices are conjectured to provide post-quantum security. We know of quantum algorithms, such as Shor's for integer factorization and Simon's for discrete log, which would break many discrete log-based (and hence pairing-based) "classical" schemes that are used in real-world applications. An important property of cryptographic systems is "forward secrecy". For example, if a large group or nation records encrypted messages of users in the present, and some time in the future large-scale quantum computers come online, those encrypted messages may become compromised. As such, so-called *post-quantum cryptography* revolving around lattice-based assumptions is relevant even to cryptographers today.

As real-world evidence of this idea, the National Institute of Standards (NIST) has standardized post-quantum key agreement protocols such as Kyber and Dilithium (among others) which are lattice-based. There is also ongoing work to standardize post-quantum signature schemes. Additionally, Apple has begun using Kyber alongside elliptic curve methods to secure iMessage messages. Cloudflare and Google also use post-quantum key agreement alongside their classical algorithms.

Another advantage of lattices is that their security is based on *worst-case* hardness. In contrast with discrete log, which is an *average-case* assumption where an adversary sees a random instance, lattice assumptions only rely on the existence of a *single* challenging instance for any particular adversarial algorithm. This is conceptually related to basing cryptography purely on the hardness of NP problems. For example, we could ask, if P $\neq$ NP, is it possible to construct cryptographic systems based on 3-SAT? Unfortunately, we don't know the answer to this, that is, we don't know how to base cryptography purely on the existence of hard problems, but lattices do seem to push us in the right direction.

Finally, lattices have nice algebraic structure and properties that give us powerful functionality. The crowning achievement to date of lattice-based cryptography is the realization of FHE, something which was previously thought to be impossible.

Definition 17.1 (Lattices)**.** An n-dimensional lattice $\mathcal{L} \subset \mathbb{R}^n$ is a discrete additive subgroup of $\mathbb{R}^n$.

- **Discrete:** For each $x \in \mathcal{L}$, there exists a neighborhood $U \subseteq \mathbb{R}^n$ about x such that $U \cap \mathcal{L} = \{x\}$.
- **Additive Subgroup:** $\forall\, x, y \in \mathcal{L}$, $x + y \in \mathcal{L}$ and $-x \in \mathcal{L}$.

A neighborhood is typically thought of as a ball about a particular point, that is, all points which are sufficiently "close" to a reference point, in terms of a norm (usually ℓ_p or ℓ_∞).

As an example, $\mathbb{Z}^2 \subset \mathbb{R}^2$ is a two-dimensional lattice. $\mathbb{Z}^2$ is clearly an additive subgroup of $\mathbb{R}^2$, and for each point $(x, y) \in \mathbb{Z}^2$, we can consider a neighborhood of points whose distance from (x, y) (in terms of ℓ_∞) norm is at most $\frac{1}{2}$. Another example: $\mathbb{Q}^n \subset \mathbb{R}^n$ is *not* a lattice. Since $\mathbb{Q}$ is dense in $\mathbb{R}$, $\mathbb{Q}^n$ is not discrete. Some other examples of lattices are $\mathbb{Z}^n$ and $(q\mathbb{Z})^n$ for $q \in \mathbb{N}$. The latter is called a q-ary lattice. We typically stray away from using visual interpretations of lattices, since we can only usually visualize one and two-dimensional lattices, but our visual intuition does not always carry over to higher dimensions.

Mathematical Properties. Lattices are typically infinite, but they can be finitely generated. We do so by writing down a basis, with elements of the lattice being expressed as integer linear combinations of basis elements. For a matrix representing a basis

$$B = \begin{bmatrix} | & | & & | \\ b_1 & b_2 & \dots & b_k \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times k},$$

we denote the lattice generated by the basis as

$$\mathcal{L}(B) := \left\{ \sum_{i=1}^{k} \alpha_i \cdot b_i \mid \alpha_i \in \mathbb{Z} \,\forall\, i \in [k] \right\}.$$

We will also sometimes write $B \cdot \mathbb{Z}^k$ to mean $\mathcal{L}(B)$. It is important that the columns $b_1, b_2, \ldots, b_k$ of B are linearly independent over $\mathbb{R}^n$, not just $\mathbb{Z}^n$. Moreover, every lattice has a basis. For example, $\mathbb{Z}^n = \mathcal{L}(I^n)$, where $I^n = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}$. A basis for a lattice generally is not unique, that is, lattices can have many bases. For example,

$$\left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \text{ and } \left\{ \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 2 \\ 1 \end{bmatrix} \right\}$$

are two different bases for $\mathbb{Z}^2$. So which basis should we use to represent a given lattice? It turns out that this is a very important question, as the hardness of many lattice problems varies greatly based on the choice of basis. Certain problems are much easier with a "good" basis than they are with a "bad" basis.

"Good" bases are those whose basis vectors have small norm (l_2 norm or l_∞ norm). The l_2 norm of a vector $v \in \mathbb{Z}^n$ is defined as $\|v\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2}$, whereas the l_∞ norm of v is defined as $\|v\|_\infty = \max_i |v_i|$. In many cases, the public parameters for lattice-based schemes will include a "bad" basis, and the secret key will be a "good" basis.

Definition 17.2 (Minimum Distance)**.** Let $\mathcal{L}$ be an n-dimensional lattice. The minimum distance of $\mathcal{L}$, $\lambda_1(\mathcal{L})$, is defined as

$$\lambda_1(\mathcal{L}) := \min_{v \in \mathcal{L} \setminus \{0\}} \|v\|.$$

Equivalently, the minimum distance is the norm of the shortest non-zero vector in $\mathcal{L}$.

Computational Problems.

- **Shortest Vector Problem (SVP):** Given a basis B for a lattice $\mathcal{L}$, find $v \in \mathcal{L}$ such that $\|v\| = \lambda_1(\mathcal{L})$.
- **Approximate SVP (SVP$_\gamma$):** Given a basis B for a lattice $\mathcal{L}$, find $v \in \mathcal{L}$ such that $\|v\| \leq \gamma \cdot \lambda_1(\mathcal{L})$.
- **Decisional approximate SVP (GapSVP$_\gamma$):** Given a basis B for a lattice $\mathcal{L}$, decided if $\lambda_1(\mathcal{L}) \leq 1$ or if $\lambda_1(\mathcal{L}) \geq \gamma$ (it is guaranteed that exactly one of these is true).

SVP is NP-hard (intuitively it would be hard to even verify if a vector was truly the shortest), but not known to be in NP (hence not NP-complete). For a reasonable value of γ, SVP$_\gamma$ is still not in NP. As such, we defer to the third and easiest problem, GapSVP$_\gamma$, which is in NP (the witness being a vector with appropriate norm). When building cryptographic schemes using lattice assumptions, we actually do use average-case assumptions, but we will see that these assumptions each reduce to the worst-case hardness of GapSVP$_\gamma$.

It seems intuitively true that GapSVP$_\gamma$ should get progressively easier as γ gets larger and larger, and it turns out that this intuition is indeed correct. The complexity of GapSVP$_\gamma$ in terms of γ is as follows:

- Between 1 and $\mathcal{O}(1)$: NP-hard (based on a randomized reduction).
- Between $\mathcal{O}(1)$ and $2^{(\log n)^{1-\varepsilon}}$: Quasi-NP-hard (based on a randomized reduction).

- $\sqrt{n/\log n}$: Both NP and coAM (based on a quasi-polynomial-time randomized reduction).
- $\sqrt{n}$: Both NP and coNP (which is the same class as problems such as factoring).
- $\mathcal{O}(n)$: Here we get an implication of one-way functions. There is a gap at which point we get PKE and FHE, etc.
- $2^{n \log\log n/\log n}$: BPP (polynomial-time).

Moreover, known GapSVP_γ algorithms run in nearly exponential time in terms of lattice dimension.

Problems used in Practice. As mentioned previously, we do not generally speaking reduce directly to GapSVP_γ when coming up with cryptographic systems. Instead, we will reduce to average-case lattice problems which themselves reduce to the worst-case hardness of GapSVP_γ. The two most common assumptions are the short integer solutions (SIS) assumption and the learning-with-errors (LWE) assumption. Here we will cover SIS, deferring the discussion of LWE to a future lecture.

Definition 17.3 ($\mathsf{SIS}_{n,m,q,\beta}$)**.** For a lattice of dimension n, a number of samples m, a modulus q, and a norm bound β, the $\mathsf{SIS}_{n,m,q,\beta}$ assumption states that there is no polynomial-time adversary $\mathcal{A}$ such that given $A \xleftarrow{R} \mathbb{Z}_q^{n\times m}$, $\mathcal{A}$ produces non-zero $x \in \mathbb{Z}_m$ such that $Ax = 0 \pmod q$ and $\|x\| \leq \beta$.

It will typically be the case that $m \gg n$. Generally, it is the case that $Ax = 0$ has plenty of solutions, but finding *short* solutions is conjectured to be hard for certain norm bounds. Setting $\beta = q/2$ makes the problem trivial (one could simply use Gaussian elimination). In particular, we care about the case where $\beta \ll q$, for example $\beta = \sqrt{q}$.

CS 395T: Topics in Cryptography March 21, 2024

Lecture 18: Short Integer Solutions

Lecturer: David Wu *Scribe: Jeriah Yu*

SIS Recap In the previous lecture, we introduced the Short Integer Solutions problem on lattices.

Definition 18.1 (Short Integer Solution Problem ($\mathsf{SIS}_{n,m,q,\beta}$))**.** The problem is defined in terms of lattice dimension n, number of instances m, modulus q, and norm bound β. Given $A \xleftarrow{R} \mathbb{Z}_q^{n\times m}$, find an $x \in \mathbb{Z}_q^m$ such that $x \neq 0$, $Ax \equiv 0 \pmod q$, and $\|x\| \leq \beta$.

We generally define $\|x\|$ to be the ℓ_2 norm, $\|x\|_2 = \sqrt{\sum_{i\in[m]} x_i^2}$. With this norm definition, we can show a solution always exists if $m > n \log q$ and $\beta \geq \sqrt{m}$.

Proof. Take $y \in \{0,1\}^m$. There are 2^m values of y, $2^m > 2^{n\log q} = q^n$. The number of possible values for $Ay \leq q^n$. Therefore, by the pigeonhole principle, there must exist $y_1 \neq y_2$ such that $Ay_1 = Ay_2$, thus we have a solution to SIS. □

SIS and SVP SIS is an average-case SVP on lattice $A \in \mathbb{Z}_q^{n\times m}$. If $\mathcal{L}^\perp(A) = \{x \in \mathbb{Z}^m : Ax \equiv 0 \pmod q\}$, then SIS is equivalent to approximate SVP in $\mathcal{L}^\perp(A)$.

Theorem 18.2. *For any $m = \mathsf{poly}(n)$, any $\beta > 0$, and sufficiently large $q \geq \beta \cdot \mathsf{poly}(n)$, there is a randomized reduction from solving GapSVP_γ in the worst case to solving $\mathsf{SIS}_{n,m,q,\beta}$ where $\gamma = \beta \cdot \mathsf{poly}(n)$.*

Inhomogeneous SIS A parallel to the SIS problem for finding x where $Ax = 0$, the ISIS problem is to find x where $Ax = b$ for a random target vector $b \in \mathbb{Z}_q^n$.

Definition 18.3 (Inhomogeneous Short Integer Solution Problem ($\mathsf{ISIS}_{n,m,q,\beta}$))**.** The problem is defined in terms of lattice dimension n, number of instances m, modulus q, and norm bound β. Given $A \xleftarrow{R} \mathbb{Z}_q^{n\times m}$ and $b \xleftarrow{R} \mathbb{Z}_q^n$, find an $x \in \mathbb{Z}_q^m$ such that $Ax \equiv b \pmod q$, and $\|x\| \leq \beta$.

Keyed Collision-Resistant Hash Functions (CRHF) from SIS Using the SIS hardness assumption, we can construct a collision resistant hash function. We recall the definition of a keyed CRHF.

Definition 18.4 (Collision-Resistant Hash Function)**.** A keyed family of hash functions $H : K \times X \to Y$, where K is the key space, X is the input space, and Y is the output space, is collision resistant if it satisfies the following properties:

- **Compression:** $|Y| < |X|$
- **Collision Resistance:** for some $k \xleftarrow{R} K$, an efficient adversary should have negligible advantage in finding $x_1 \neq x_2 \in X$ such that $H(k, x_1) = H(k, x_2)$.

We can derive a CRHF directly from SIS. For $H : \mathbb{Z}_q^{n\times m} \times \{0,1\}^m \to \mathbb{Z}_q^n$, randomly sample $A \xleftarrow{R} K$, and define $H(A, x) = Ax \pmod q$. The compression property is satisfied by setting $m > n \log q$, and the collision resistance property follows directly from SIS: suppose for $A \xleftarrow{R} \mathbb{Z}_q^{n\times m}$ an efficient adversary outputs $x_1 \neq x_2 \in \{0,1\}^m$ s.t. $H(A, x_1) = H(A, x_2)$. Then $A(x_1 - x_2) = 0$, thus solving SIS.

Local Updates For most cryptographic hash functions such as Merkle-Damgard constructions, changing a small part of the input requires rerunning the hash function on the entire input. For the SIS CRHF, this is unnecessary for small changes to x.

Suppose $y = H(A, x)$ where $x \in \{0,1\}^m$. Let x be updated at a single bit index i^* from x_{i^*} to x'_{i^*}. Since $H(A, x) = Ax = \sum_{i \in [m]} a_i x_i$, where a_i are the column vectors of A, we can easily find the updated hash by adding $a_{i^*} \cdot (x'_{i^*} - x_{i^*}$ to the original hash.

Universality and Leftover Hash Lemma

Definition 18.5 (Universality)**.** Let $H : K \times X \to Y$ be a keyed hash function. H is universal if for all $x_0 \neq x_1$, $\Pr[H(k, x_0) = H(k, x_1)] \leq \frac{1}{|Y|}$ for a randomly sampled key k.

This is an information-theoretic property, and does not rely on computational assumptions of regarding efficient adversaries.

Theorem 18.6. *The SIS CRHF is universal.*

Proof. Let $x_1 \neq x_2 \in \{0,1\}^m$ and $A \xleftarrow{R} \mathbb{Z}_q^{n \times m}$. $Ax_1 = Ax_2 \implies A(x_1 - x_2) = \sum_{i \in [m]} a_i(x_{1,i} - x_{2,i}) = 0$. Since $x_1 \neq x_2$, there must be some index i^* where they differ: $x_{1,i^*} \neq x_{2,i^*} \implies (x_{1,i^*} - x_{2,i^*}) \in \{-1, 1\}$. Then $Pr[Ax_1 = Ax_2 \pmod q] = Pr[a_{i^*} = -(x_{1,i^*} - x_{2,i^*})^{-1} \cdot \sum_{i \neq i^*} a_i(x_{1,i} - x_{2,i})] = \frac{1}{q^n} = \frac{1}{|Y|}$. Thus, the SIS CRHF is universal by definition. □

We now define guessing probability and min-entropy.

Definition 18.7 (Guessing Probability and Min-entropy)**.** Let X be a random variable which takes values from a finite set S. The guessing probability of X is $\max_{s \in S} Pr[X = s]$. The min-entropy of X is $-\log$ guessing probability, or $-\log(\max_{s \in S} Pr[X = s])$. Essentially, a random variable with k bits of min-entropy can be treated as a random bit string of length k.

Definition 18.8 (Leftover Hash Lemma)**.** Let $H : K \times X \to Y$ be a universal hash. Let $x \in X$ be a random variable with t bits of min-entropy. Define distributions:

- $D_0 := (k \xleftarrow{R} K, y \leftarrow H(k, x))$
- $D_1 := (k \xleftarrow{R} K, y \xleftarrow{R} Y)$

Then the statistical distance between D_0 and D_1 is bounded by $\frac{1}{2}\sqrt{\frac{|Y|}{2^t}}$.

For security parameter λ, we want to set $t = 2\lambda + \log |Y|$ to achieve a statistical distance $< \frac{1}{2^\lambda}$.

Note that statistical distance is different than computational advantage. Any adversary, without any efficiency requirement, cannot distinguish between D_0 and D_1 with advantage greater than ϵ if the statistical distance between D_0 and D_1 is ϵ.

For lattices, m is commonly set to be $3n \log q$ as this achieves negligible distinguishing advantage against uniform random sampling.

Claim 18.9. *Let $A \xleftarrow{R} \mathbb{Z}_q^{n \times m}$ where $m > 3n \log q$. Then for any $r \xleftarrow{R} \{0,1\}^m$, Ar is statistically close to uniform over $\mathbb{Z}_q^n$.*

Proof. $|Y| = |\mathbb{Z}_q^n| = q^n$. The statistical distance is $\sqrt{\frac{|Y|}{2^t}} = \sqrt{\frac{q^n}{2^{3n \log q}}} = \sqrt{\frac{q^n}{(q^n)^3}} = \frac{1}{q^n}$. This is the equivalent probability to uniformly sampling $\mathbb{Z}_q^n$. □

Commitment from LHL

Definition 18.10 (Commitment)**.** Recall the definition of commitment schemes using three functions: Setup, Commit, and Verify.

- $\mathsf{Setup}(1^\lambda) \to crs$: create a common reference string relative to the security parameter.
- $\mathsf{Commit}(crs, m, r) \to c$: commit to message m using random element r.
- $\mathsf{Verify}(crs, c, r, m) \to 0/1$: check if c is a valid commitment to m with opening r.

A secure commitment scheme should satisfy the following two properties:

- **Hiding**: Adversaries have negligible advantage distinguishing between commitments of different messages.
- **Binding**: For a random crs, adversaries have negligible advantage in computing (m_1, r_1) and (m_2, r_2) where $m_1 \neq m_2$ such that $c_1 = c_2$.

Definition 18.11 (Pedersen commitment from discrete log)**.** Recall the Pedersen commitment using discrete log:

- Setup: sample $s \xleftarrow{R} \mathbb{Z}_p$, output $(g, h = g^s)$.
- $\mathsf{Commit}((g, h), m)$: sample $r \xleftarrow{R} \mathbb{Z}_p$, output $c = g^r h^m$, with opening r.
- $\mathsf{Verify}((g, h), c, r, m)$: check if $c \stackrel{?}{=} g^r h^m$.
- **Hiding**: the Pedersen commitment satisfies the hiding property as g^r is indistinguishable from a uniform random group element and obscures h^m.
- **Binding**: the binding property is satisfied as an adversary that finds $g^r h^m = g^{r'} h^{m'}$ with $m \neq m'$ can solve the discrete log problem by solving $s = (r - r')(m - m')^{-1} \bmod q$.

We can construct a parallel commitment scheme from SIS.

Definition 18.12 (Commitment from SIS)**.** Let n, q be lattice parameters and $m > 3n \log q$. We define the following commitment functions:

- Setup: $A_1 \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $A_2 \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, output $crs = (A_1, A_2)$
- $\mathsf{Commit}((A_1, A_2), p \in \{0,1\}^m)$: sample $r \xleftarrow{R} \{0,1\}^m$, output $c = \left[A_1 \;\middle|\; A_2\right] \left[\begin{array}{c} p \\ \hline r \end{array}\right] = A_1 p + A_2 r$
- $\mathsf{Verify}((A_1, A_2), c, r, p))$: check $p, r \in \{0,1\}^m$ and $c \stackrel{?}{=} A_1 p + A_2 r$

We can now prove the properties for commitment schemes:

- **Hiding**: since $m > 3n \log q$, $A_2 r$ is statistically indistinguishable from a uniform random value in $\mathbb{Z}_q^n$ by the Leftover Hash Lemma. It therefore acts as a blinding element obscuring $A_1 m$.

- **Binding**: Assume there exists an efficient adversary that can break the binding property. It can be used to construct an adversary to solve $\mathsf{SIS}_{n,2m,q,\sqrt{2m}}$:

  1. SIS challenger samples $A = [A_1|A_2] \xleftarrow{R} \mathbb{Z}_q^{n\times 2m}$ and sends to SIS adversary.
  2. The SIS adversary provides $crs = (A_1, A_2)$ to binding adversary.
  3. The binding adversary responds with $(m_1, r_1), (m_2, r_2)$ such that $A_1 m_1 + A_2 r_1 = A_1 m_2 + A_2 r_2$.
     Or alternatively, $A_1(m_1 - m_2) + A_2(r_1 - r_2) = A \begin{bmatrix} m_1 - m_2 \\ r_1 - r_2 \end{bmatrix} = 0$
  4. The SIS adversary outputs $x = \begin{bmatrix} m_1 - m_2 \\ r_1 - r_2 \end{bmatrix}$

  If the binding adversary succeeds, we show above that the SIS adversary succeeds in finding x such that $Ax = 0$, and $x \neq 0$ since $m_1 \neq m_2$. Since $x \in \{-1, 0, 1\}^{2m}$, $\|x\|_2 \leq \sqrt{2m}$. Thus, x is a valid solution to the SIS problem.

CS 395T: Topics in Cryptography March 26, 2024

Lecture 19: Lattice Trapdoors

Lecturer: David Wu *Scribe: Jeriah Yu*

Lattice Trapdoors from ISIS Recall the definition of Inhomogeneous SIS:

Definition 19.1 (Inhomogeneous Short Integer Solution Problem ($\mathsf{ISIS}_{n,m,q,\beta}$))**.** The problem is defined in terms of lattice dimension n, number of instances m, modulus q, and norm bound β. Given $A \xleftarrow{R} \mathbb{Z}_q^{n\times m}$ and $b \xleftarrow{R} \mathbb{Z}_q^n$, find an $x \in \mathbb{Z}_q^m$ such that $Ax \equiv b \pmod q$, and $\|x\| \le \beta$.

Our eventual goal is to create digital signatures in lattices. To accomplish this, we first need to create trapdoor functions using the ISIS problem.

Definition 19.2 (Trapdoor Function)**.** A trapdoor function is a function f that is easy to compute in one direction but hard to compute an inverse unless given additional information. This information is called the trapdoor. $f(x) \to y$ is easy, $f^{-1}(y) \to x$ is hard, $f^{-1}(y) \xrightarrow{td} x$ is easy.

Definition 19.3 (Lattice Trapdoor)**.** A lattice trapdoor is defined with the following procedures:

- $\mathsf{TrapGen}(n, m, q, \beta) \to (A, td_A)$: given lattice parameters, the algorithm outputs matrix A and trapdoor td_A, where $A \in \mathbb{Z}_q^{n\times m}$. A should be statistically close to uniform over $\mathbb{Z}_q^{n\times m}$.
- $f_A(x)$: for input $x \in \mathbb{Z}_q^m$, output Ax.
- $f_A^{-1}(td_A, y)$: for input $y \in \mathbb{Z}_q^n$, output x such that $Ax = y$ and $\|x\| \le \beta$.

To make a trapdoor for the ISIS problem, we will make use of a powers-of-two "gadget" matrix.

Definition 19.4 (Gadget Matrix)**.** The gadget matrix $G \in \mathbb{Z}_q^{n\times n\lfloor \log q\rfloor}$ is a sparse matrix, where $\ell = \lfloor \log q \rfloor$:

$$G = \begin{bmatrix} 1 & 2 & 4 & \dots & 2^\ell & & & & & \\ & & & & & 1 & 2 & 4 & \dots & 2^\ell \\ & & & & & & & & & \ddots \end{bmatrix}$$

This is equivalent to the tensor product $\mathbf{I}_n \otimes \begin{pmatrix} 1 & 2 & 4 & \dots & 2^\ell \end{pmatrix}$, where $\mathbf{I}_n$ is the identity matrix. Since generally $m > n\lfloor \log q \rfloor$, G can be padded with columns of 0 to achieve $G \in \mathbb{Z}_q^{n\times m}$.

Inhomogeneous SIS is easy if $A = G$, as x can be set to the bitwise binary decomposition of y. This yields $Gx = y$, and since x only contains 0 and 1, $\|x\| = 1$ for the infinity norm. We can refer to the bit decomposition function as $G^{-1} : \mathbb{Z}_q^n \to \{0,1\}^m$.

We now want to hide G in relation to A in the ISIS problem, which is uniformly sampled. Thus, our secret trapdoor for a specific instance of ISIS should be a "short" matrix with small entries R such that $AR = G$. Assuming we have this matrix, we can define $f_A^{-1}(td_A = R, y)$: output$x = RG^{-1}(y)$. Then we see that $Ax = ARG^{-1}(y) = GG^{-1}(y) = y$. Then since $\|G^{-1}(y)\| = 1$, then $\|x\| \le m \cdot \|R\|$, so if $\|R\|$ is small enough, we can ensure $\|x\| \le \beta$.

How do we find A and R such that $AR = G$, A is statistically close to uniform, and R is short? We can define $\mathsf{TrapGen}(n, m, q, \beta)$ as follows:

1. Sample $\bar{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}, \bar{R} \xleftarrow{R} \{0,1\}^{m \times m}$.
2. Compute $A = \begin{bmatrix} \bar{A} & | & \bar{A}\bar{R} + G \end{bmatrix}$ and $R = \begin{bmatrix} -\bar{R} \\ I_m \end{bmatrix}$
3. Output A and $td_A = R$.

By construction, $AR = G$ and $\|R\| = 1$. By Leftover Hash Lemma, if $m \geq 3n \log q$, then Ar where $r \xleftarrow{R} \{0,1\}^m$ is statistically close to a uniform element of $\mathbb{Z}_q^n$ given A. We repeat this claim for each column of $\bar{R}$ to show that $\bar{A}\bar{R}$ is statistically close to uniform in $\mathbb{Z}_q^{n \times m}$ given $\bar{A}$, thus A is statistically close to uniform in $\mathbb{Z}_q^{n \times 2m}$.

Digital Signatures from Lattice Trapdoors With lattice trapdoors, we can develop a digital signature scheme in the random oracle model that is analogous to RSA.

We start by defining a candidate signature scheme using the gadget trapdoor defined above:

- $\mathsf{KeyGen}(1^\lambda) : (A, td_A = R) \leftarrow \mathsf{TrapGen}(n, m, q, \beta)$. Output $vk = A, sk = td_A = R$ where $AR = G$.
- $\mathsf{Sign}(sk, m) : \sigma \leftarrow f_A^{-1}(sk, H(m)) = sk \cdot G^{-1}(H(m))$, where $H(m) : \{0,1\}^* \to \mathbb{Z}_q^n$ is a random oracle hash.
- $\mathsf{Verify}(vk, m\sigma)$: Check $\|\sigma\| \leq \beta$ and $f_A(\sigma) = A\sigma = H(m)$.

We can see that this signature is correct by inspection. With regards to security, a forgery requires the adversary to find v such that $Av = H(m)$. Since A is statistically close to uniform and $H(m)$ is uniform, this is equivalent to solving ISIS. However, in the security reduction, we need to answer signing queries for the signature adversary. Every signing query leaks information about R. This is because $H(m) = y$ and $G^{-1}(y)$ are both computable. Thus an adversary can reconstruct R by making multiple signing queries to construct and solve a linear system.

To remedy this issue, we introduce randomness into the signing algorithm using a preimage sampleable trapdoor function.

Definition 19.5. A function $f : X \to Y$ is a preimage-sampleable trapdoor function if there exists an efficiently-sampleable distribution D over X and a trapdoor inversion algorithm $\mathsf{SamplePre}$ with the following properties:

- For $x \leftarrow D, f(x)$ is uniform in Y.
- For $y \xleftarrow{R} Y$, $\mathsf{SamplePre}(td, y)$ outputs a preimage x from D where $f(x) = y$.
- $\{x \leftarrow D, y \leftarrow f(x) : (x, y)\}$ (a forward sampling) is statistically close to $\{y \xleftarrow{R} Y, x \leftarrow \mathsf{SamplePre}(td, y) : (x, y)$ (a backward sampling).
- Given f and $y \xleftarrow{R} Y$, an efficient adversary has negligible advantage in computing x such that $F(x) = y$ without td.

Replacing the gadget trapdoor G^{-1} with a preimage-sampleable trapdoor $\mathsf{SamplePre}$ allows us to develop a security proof analogous to RSA-FDH.

Proof. We will construct adversary B to break the trapdoor function security given an efficient signature adversary A. Without loss of generality, assume A queries the random oracle H on message m before making a signing query on m.

1. Adversary B receives $y^* \xleftarrow{R} Y$ from the challenger, and randomly chooses a query i^* to program y^* to.
2. For a random oracle query, respond with $y \leftarrow y^*$ for query i^*, otherwise forward sample and add $m \mapsto (x, y)$ to the random oracle table.
3. For a signing query, if m has an entry in the table, respond with x. Otherwise, abort.
4. Adversary A completes with m^*, σ^*. If m^* is the message used in random oracle query i^*, output σ^*. Otherwise, abort.

If adversary A makes Q random oracle queries and has non-negligible advantage ϵ, B succeeds with advantage $\frac{\epsilon}{Q}$, which is non-negligible because Q is polynomially-bounded since A is efficient. Random oracle queries are properly distributed by preimage sampling and signature queries are properly distributed, so if A succeeds and i^* is a correct guess, $f(\sigma^*) = H(m^*) = y^*$, allowing B to break the one-wayness of the trapdoor function. □

Preimage-Sampleable Trapdoor from SIS Recall the hash function from SIS: $f_A(x) = Ax \pmod q$, where $A \in \mathbb{Z}_q^{n \times m}$ and $x \in \mathbb{Z}_q^m$. We wish to construct a preimage-sampelable trapdoor, which requires a sufficient distribution on $\mathbb{Z}_q^m$ to sample preimages. For lattices, we use the discrete Gaussian distribution.

Definition 19.6 (Discrete Gaussian Distribution)**.** Define the mass function $\rho_s(x) = \exp\left(\frac{-\pi \|x\|_2^2}{s^2}\right)$, with width s. Then the discrete Gaussian distribution $D_{\mathbb{Z}^m, s}$ over $\mathbb{Z}_m$ is the distribution with probability mass function $Pr[x = z] = \frac{\rho_s(z)}{\sum_{x \in \mathbb{Z}^m} \rho_s(x)} \forall z \in \mathbb{Z}^m$.

For $A \in \mathbb{Z}_q^{n \times m}$ and $y \in \mathbb{Z}_q^m$, we will denote $x \leftarrow A_s^{-1}(y)$ as the distribution $x \leftarrow D_{\mathbb{Z}^m, s}$ conditioned on $Ax = y$. Now we want to ensure we can perform efficient preimage-sampling on this distribution.

Theorem 19.7. *Suppose $AR = G$ and $s \geq \|R\|_\infty \cdot \log n$. Then there exists an efficient algorithm* SamplePre *such that the distributions $\{x \leftarrow \mathsf{SamplePre}(A, R, y, s)\}$ and $\{x \leftarrow A_s^{-1}(y)\}$ are statistically close (within 2^{-n}) for all $y \in \mathbb{Z}_q^n$.*

If we ensure that $m \geq 2n \log q$, $s \geq \log m$, $A \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, and $x \leftarrow D_{\mathbb{Z}^m, s}$, Ax will be statistically close to uniform.

CS 395T: Topics in Cryptography March 28, 2024

Lecture 20: Lattice-based Signatures and Learning with Errors (LWE)

Lecturer: David Wu *Scribe: Sidh Suchdev*

Lattice-based Signatures Recap Recall from last lecture the definition of a preimage-sampleable trapdoor function:

Definition 20.1. A function $f\colon X \to Y$ is a **preimage-sampleable trapdoor function** if there exists an efficiently-sampleable distribution $\mathcal{D}$ and an inversion algorithm $\mathsf{SamplePre}$ where:

$$\left\{ \begin{array}{c} x \leftarrow \mathcal{D} \\ y \leftarrow f(x) \end{array} : (x, y) \right\} \overset{s}{\approx} \left\{ \begin{array}{c} y \xleftarrow{R} Y \\ x \leftarrow \mathsf{SamplePre}(\mathsf{td}, x) \end{array} : (x, y) \right\}$$

We define the preimage sampling theorem for the ISIS trapdoor:

Theorem 20.2. *Let $A \in \mathbb{Z}_q^{n \times m}$, R small, where $AR = G$ and $s \geq \|R\|_\infty \cdot \log n$. Then there exists an efficient algorithm* $\mathsf{SamplePre}$ *such that*

$$\{x \leftarrow \mathsf{SamplePre}(A, R, y, s)\} \overset{s}{\approx} \{x \leftarrow A_s^{-1}(y)\}$$

for all $y \in \mathbb{Z}_q^n$.

Corollary 20.3. *Using the preimage-sampling theorem, we can see that:*

$$\left\{ \begin{array}{c} (A, R) \leftarrow \mathsf{TrapGen} \\ x \leftarrow D_{\mathbb{Z},s} \end{array} : (A, x, Ax) \right\} \overset{s}{\approx} \left\{ \begin{array}{c} (A, R) \leftarrow \mathsf{TrapGen} \\ y \xleftarrow{R} \mathbb{Z}_q^m \\ x \leftarrow f_A^{-1}(R, y) \end{array} : (A, x, y) \right\}$$

Learning With Errors (LWE) Learning with Errors, like SIS, is a fundamental assumption in lattice-based cryptography. It can be viewed as the dual to the Diffie-Hellman assumption for lattice-based cryptography.

Definition 20.4 (LWE assumption)**.** The $\mathsf{LWE}_{n,m,q,\chi}$ assumption states that:

$$(A, s^\intercal A + e^\intercal) \overset{c}{\approx} (A, r^\intercal),$$

where $A \xleftarrow{R} \mathbb{Z}_q^{n \times m}, s \xleftarrow{R} \mathbb{Z}_q^n, e \xleftarrow{R} \chi^m, r \xleftarrow{R} \mathbb{Z}_q^m$. LWE is parameterized by n, m, q, χ. Typical parameter values are:

- $n \approx 1024$
- $q \approx 2^{32}$
- $m \sim n \log q \approx 2^{15}$
- $\chi = D_{\mathbb{Z},s}$ where $s \approx 3$

It is not obvious that this assumption is hard. However, Regev showed that LWE reduces to GapSVP using a quantum reduction. More formally, he showed that for any $m = \mathsf{poly}(n)$, modulus $q < 2^{\mathsf{poly}(n)}$ and $\chi = D_{\mathbb{Z},s}$ where χ is β-bounded ($\Pr[x \leftarrow \chi \colon |x| < \beta] = 1$), then solving $\mathsf{LWE}_{n,m,q,\chi}$ is as hard as solving GapSVP_{γ}, where $\gamma = \tilde{\mathcal{O}}(n \cdot \frac{q}{\beta})$.

Additionally, it can be shown that the hardness of LWE implies the hardness of SIS. We can construct an informal proof showing this. Suppose we have an adversary $\mathcal{A}$ for the SIS problem. We will construct an adversary $\mathcal{B}$ for the LWE game.

1. The challenger gives $\mathcal{B}$ an LWE challenge $(A, t^\intercal)$. $\mathcal{B}$ must decide whether $t^\intercal \stackrel{?}{=} s^\intercal A + e^\intercal$ ($b = 0$) or if $t^\intercal \stackrel{?}{=} u^\intercal$ where $u \xleftarrow{R} \mathbb{Z}_q^m$ ($b = 1$).
2. $\mathcal{B}$ sends A to $\mathcal{A}$ and receives a vector $z \in \mathbb{Z}^m$ such that $Az = 0 \mod q$ and $||z||$ is "small".
3. $\mathcal{B}$ computes $t^\intercal \cdot z$.
   - If $t^\intercal = s^\intercal A + e^\intercal$, then $t^\intercal \cdot z = s^\intercal A \cdot z + e^\intercal \cdot z = e^\intercal \cdot z$. Observe that $e^\intercal \cdot z$ is "small" since $e \in \chi^m$ and z are both "small".
   - If $t^\intercal = u^\intercal$, then $t^\intercal \cdot z = u^\intercal \cdot z$. Observe this value is likely "large" since u is sampled uniformly randomly from $\mathbb{Z}_q$.
4. $\mathcal{B}$ sends the guess $b' = 0$ if $||t^\intercal \cdot z||$ is "small" and $b' = 1$ otherwise.

Definition 20.5 (Regev Symmetric Encryption Scheme)**.** The definition of the scheme follows below:

- $\mathsf{Setup}(1^\lambda)$: Output $s \xleftarrow{R} \mathbb{Z}_q^n$
- $\mathsf{Encrypt}(s, \mu \in \{0,1\})$: Sample $a \xleftarrow{R} \mathbb{Z}_q^n$ and $e \xleftarrow{R} \chi$. Output $\mathsf{ct} = (a, s^\intercal a + e + \mu \lfloor \frac{q}{2} \rfloor)$
- $\mathsf{Decrypt}(s, \mathsf{ct} = (c_1, c_2))$: Output $\lfloor c_2 - s^\intercal c_1 \pmod q \rceil_{q,2}$, where $\lfloor \cdot \rceil$ is the rounding operation and $\lfloor x \rceil_{q,2} \coloneqq \left\lfloor \frac{2}{q} \cdot x \right\rceil$

Correctness Let $\mathsf{ct} = (c_1, c_2) \leftarrow \mathsf{Encrypt}(s, \mu)$. It follows that

$$\begin{aligned}
\lfloor c_2 - s^\intercal c_1 \pmod q \rceil_{q,2} &= \left\lfloor s^\intercal a + e + \mu \left\lfloor \frac{q}{2} \right\rfloor - s^\intercal a \pmod q \right\rceil_{q,2} \\
&= \left\lfloor e + \mu \left\lfloor \frac{q}{2} \right\rfloor \pmod q \right\rceil_{q,2} \\
&\approx \left\lfloor \frac{2}{q} e + \mu \pmod q \right\rceil
\end{aligned}$$

Note that if $|e| < \frac{q}{4}$, then we can see that the result of the rounding operation is unchanged by e, and thus rounds to the value of μ.

Security We can see security directly from the LWE assumption. We will show each encryption is computationally indistinguishable from a uniformly random sample:

- $\mathsf{Encrypt}(s, 0) = (a, s^\mathsf{T} a + e) \overset{c}{\approx} (a, r)$
- $\mathsf{Encrypt}(s, 1) = (a, s^\mathsf{T} a + e + \lfloor \frac{q}{2} \rfloor) \overset{c}{\approx} (a, r + \lfloor \frac{q}{2} \rfloor) \overset{c}{\approx} (a, r)$

CS 395T: Topics in Cryptography April 2, 2024

Lecture 21: Regev Encryption and Fully Homomorphic Encryption

Lecturer: David Wu *Scribe: Eli Bradley*

In the previous lecture, we introduced the Regev symmetric-key encryption scheme. In this lecture, we show the scheme can be extended to a public-key encryption scheme [Reg05] and a fully homomorphic encryption (FHE) scheme [GSW13], and introduce the intermediate notion of somewhat homomorphic encryption (SWHE).

Notation 21.1. We will use $\lfloor \cdot \rceil : \mathbb{R} \to \mathbb{Z}$ to denote rounding, and define $\lfloor x \rceil_{q,2} : \left\lfloor \frac{2}{q} x \right\rceil$. When x is a vector, this applies elementwise.

Regev encryption. Recall the Regev encryption scheme:

- $\mathsf{Setup}(1^\lambda)$: Sample and output $s \xleftarrow{R} \mathbb{Z}_q^n$.
- $\mathsf{Encrypt}(s, \mu \in \{0,1\})$: Sample $a \xleftarrow{R} \mathbb{Z}_q^n$, $e \leftarrow \chi$. Output $\mathsf{ct} = (a, s^T a + e + \mu \left\lfloor \frac{q}{2} \right\rfloor)$.
- $\mathsf{Decrypt}(s, (c_0, c_1))$: Output $\lfloor c_1 - s^T c_0 \pmod q \rceil_{q,2}$.

Correctness. Let $\mathsf{ct} = (c_1, c_0) = \mathsf{Encrypt}(s, \mu) = (a, s^T a + e + \mu \left\lfloor \frac{q}{2} \right\rfloor)$. If $|e| < \frac{q}{4}$, then $c_1 - s^T c_0 = e + \mu \left\lfloor \frac{q}{2} \right\rfloor$ rounds to μ, and the decrypted value $\mathsf{Decrypt}(s, \mathsf{ct})$ is correct.

Security. The scheme is CPA-secure assuming the hardness of the LWE problem, which we recall here.

Definition 21.2 (LWE Problem)**.** Given $a \xleftarrow{R} \mathbb{Z}_q^n, e \leftarrow \chi, s \xleftarrow{R} \mathbb{Z}_q^n, u \xleftarrow{R} \mathbb{Z}_q$, distinguish $(a, s^T a + e)$ and (a, u).

By this assumption, in the security argument, we can replace the ciphertexts with $(a, u + \mu \lfloor \frac{q}{2} \rfloor)$. Then, the encryption scheme becomes a one-time pad.

Extending the Regev encryption scheme to a public-key encryption scheme will rely on the observation that it is additively homomorphic. That is, for ciphertexts $\mathsf{ct}_1 \leftarrow \mathsf{Encrypt}(s, \mu_1)$ and $\mathsf{ct}_2 \leftarrow \mathsf{Encrypt}(s, \mu_2)$, $\mathsf{ct}_1 + \mathsf{ct}_2$ is an encryption of $\mu_1 + \mu_2$. In fact, these techniques will work to derive public-key encryption from any additively homomorphic secret-key encryption scheme.

Showing additive homomorphism. To see that Regev encryption is additively homomorphic, consider two ciphertexts:

$$\mathsf{ct}_1 = (a_1, s^T a_1 + e_1 + \mu_1 \left\lfloor \frac{q}{2} \right\rfloor).$$
$$\mathsf{ct}_2 = (a_2, s^T a_2 + e_2 + \mu_2 \left\lfloor \frac{q}{2} \right\rfloor).$$

Their sum is the following.

$$\mathsf{ct}_1 + \mathsf{ct}_2 = \left(a_1 + a_2, s^T(a_1 + a_2) + (e_1 + e_2) + (\mu_1 + \mu_2) \left\lfloor \frac{q}{2} \right\rfloor\right).$$

Assuming small enough errors, it follows that $\mathsf{ct}_1 + \mathsf{ct}_2$ is an encryption of $\mu_1 + \mu_2 \pmod 2$ if $\mu_1 = 0$ or $\mu_2 = 0$. If $\mu_1 = \mu_2 = 1$, we observe $(\mu_1 + \mu_2) \left\lfloor \frac{q}{2} \right\rfloor = 2 \left\lfloor \frac{q}{2} \right\rfloor \in \{q-1, q\} = \{-1, 0\}$. This is small and can be added into the error term, so $\mathsf{ct}_1 + \mathsf{ct}_2$ is an encryption of $\mu_1 + \mu_2 \pmod 2$ in this case as well.

Regev public-key encryption [Reg05]. In the public-key encryption scheme, the public key will consist of many encryptions of 0, $(a_i, s^T a_i + e_i)$ for $i \in [m]$. Notice this can be equivalently represented more compactly by matrices, $(A, s^T A + e^T)$. To get a fresh encryption of 0, we take the sum of a random subset of the encryptions of 0 in the public key. Then, we get an encryption of μ from this by adding $\mu \left\lfloor \frac{q}{2} \right\rfloor$ to the second component.

- $\mathsf{Setup}(1^\lambda)$: Sample $A \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $s \xleftarrow{R} \mathbb{Z}_q^n$, $e \leftarrow \chi^m$. Output $\mathsf{pk} = (A, s^T A + e^T)$, $\mathsf{sk} = s$.
- $\mathsf{Encrypt}(\mathsf{pk}, \mu \in \{0,1\})$: Parse $\mathsf{pk} = (A, b^T)$. Sample $r \xleftarrow{R} \{0,1\}^m$. Output $\mathsf{ct} = (Ar, b^T r + \mu \left\lfloor \frac{q}{2} \right\rfloor)$.
- $\mathsf{Decrypt}(s, (c_0, c_1))$: Output $\lfloor c_1 - s^T c_0 \pmod q \rceil_{q,2}$.

Correctness. We will need the following notion.

Definition 21.3 (B-bounded)**.** A distribution χ is B-bounded if

$$\Pr[x \leftarrow \chi : |x| \leq B] = 1.$$

Assume χ is B-bounded and $m \cdot B \leq \frac{q}{4}$. Let $\mathsf{ct} = (c_1, c_0) = \mathsf{Encrypt}(s, \mu) = (Ar, b^T r + \mu \left\lfloor \frac{q}{2} \right\rfloor)$. Then, $c_1 - s^T c_0 = e^T r + \mu \left\lfloor \frac{q}{2} \right\rfloor$. Notice $|e^T r| \leq m \cdot B$. If $m \cdot B \leq \frac{q}{4}$, then $c_1 - s^T c_0$ rounds to μ, and the decrypted value $\mathsf{Decrypt}(s, \mathsf{ct})$ is correct.

Notice $m \cdot B$ grows with the dimension m. It is important in LWE-based constructions for this reason that errors are short. This is also why Gaussian distributions, which have vanishingly small tails, are often used.

Security. Our security argument proceeds in two steps.

1. Note that under the LWE assumption, the public key is indistinguishable from random.

$$\mathsf{pk} \overset{c}{\approx} (A, b^T) : \begin{matrix} A \xleftarrow{R} \mathbb{Z}_q^{n \times m} \\ b \xleftarrow{R} \mathbb{Z}_q^m \end{matrix}$$

2. The ciphertexts take the following form.

$$\mathsf{ct} = \begin{bmatrix} A \\ b^T \end{bmatrix} r + \begin{bmatrix} 0 \\ \mu \cdot \left\lfloor \frac{q}{2} \right\rfloor \end{bmatrix}.$$

   But $\begin{bmatrix} A \\ b^T \end{bmatrix}$ is a uniformly random matrix and r is a random binary vector. By the leftover hash lemma, if $m \geq 3n \log q$, the first term $\begin{bmatrix} A \\ b^T \end{bmatrix} r$ is statistically close to uniformly random. The second term involving μ is hidden as in a one-time pad.

This procedure can be applied to get PKE from any additively homomorphic SKE scheme.

Fully homomorphic encryption. A fully homomorphic encryption (FHE) scheme is a scheme supporting arbitrary computation on ciphertexts. In other words, there exists a public algorithm for which for any efficiently computable function f,

$$\mathsf{Encrypt}(\mathsf{pk}, x) \xrightarrow{\text{public algorithm}} \mathsf{Encrypt}(\mathsf{pk}, f(x)).$$

This notion goes back to the beginning of cryptography, and was speculated to exist by Whitfield Diffie and Martin Helman. There were several partial successes before it was achieved.

- ElGamal supported additively homomorphic encryption, but only addition. See lecture 6.
- Boneh-Goh-Nissim supported an arbitrary number of additions, but only 1 multiplication [BGN05]. This is a pairings-based construction. See lecture 11.

Finally, Gentry showed in a 2009 PhD thesis that fully homomorphic encryption is possible [Gen09]. This was a surprising breakthrough result, and turned lattice-based cryptography from a niche field into something very popular. This construction was initially very complicated, but was significantly simplified by 2013 [GSW13].

Gentry's approach takes the following general blueprint:

1. Build somewhat homomorphic encryption (SWHE), which supports many additions and multiplications but not arbitrarily many.
2. Bootstrap from SWHE to FHE.

In this lecture, we fully cover the first step and briefly describe the second.

Gentry-Sahai-Waters (GSW) somewhat homomorphic encryption [GSW13]. We will start by presenting again the Regev public-key encryption scheme described above, slightly reframed.

- $\mathsf{Setup}(1^\lambda)$: Sample $\overline{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\overline{s} \xleftarrow{R} \mathbb{Z}_q^n$, $e \leftarrow \chi^m$.
  Output $\mathsf{pk} = A = \begin{bmatrix} \overline{A} \\ \overline{s}^T \overline{A} + e^T \end{bmatrix}$, $\mathsf{sk} = s$ where $s^T = \begin{bmatrix} -\overline{s}^T & 1 \end{bmatrix}$.
- $\mathsf{Encrypt}(A, \mu \in \{0,1\})$: Sample $r \xleftarrow{R} \{0,1\}^m$. Output $c = Ar + \begin{bmatrix} 0 \\ \mu \left\lfloor \frac{q}{2} \right\rfloor \end{bmatrix}$.
- $\mathsf{Decrypt}(s, c)$: Output $\lfloor s^T c \rceil_{q,2}$.

Note $s^T A = -\overline{s}^T \overline{A} + \overline{s}^T A + e^T = e^T$. Let $c = \mathsf{Encrypt}(A, \mu)$ and consider the output $\mathsf{Decrypt}(s, c)$.

$$\begin{aligned} s^T c &= s^T \left(Ar + \begin{bmatrix} 0 \\ \left\lfloor \frac{\mu}{2} \right\rfloor \end{bmatrix} \right) \\ &= e^T r + \begin{bmatrix} -\overline{s}^T & 1 \end{bmatrix} \begin{bmatrix} 0 \\ \left\lfloor \frac{\mu}{2} \right\rfloor \end{bmatrix} \\ &= e^T r + \mu \left\lfloor \frac{\mu}{2} \right\rfloor. \end{aligned}$$

The encryption scheme works as before.

For simplicity, we will write μ in place of $\mu \left\lfloor \frac{q}{2} \right\rfloor$ for the rest of this section. To additionally get a multiplicative homomorphism, we will try to make it so that for ciphertext matrix C, $s^T C = \mu s^T + \text{error}$ for some error term. Suppose we update the encryption scheme as follows:

- $\mathsf{Encrypt}(A, \mu \in \{0,1\})$: Sample $R \xleftarrow{R} \{0,1\}^{m \times m}$. Output $C = AR + \mu I$.

This meets the desired property: $s^T C = \mu s^T (AR + \mu I) = \mu s^T + e^T R$.

We will check for additive and multiplicative homomorphisms. Suppose we have ciphertexts C_1, C_2 of values μ_1, μ_2, so $s^T C_1 = \mu_1 s^T + e_1^T$ and $s^T C_2 = \mu_2 s^T + e_2^T$. Additive homomorphism follows by linearity as before. In addition, the following holds:

$$\begin{aligned} s^T C_1 C_2 &= (\mu_1 s^T + e_1^T) C_2 \\ &= \mu_1 s^T C_2 + e_1^T C_2 \\ &= \mu_1 \mu_2 s^T + (\mu_1 e_2^T + e_1^T C_2) \end{aligned}$$

This almost works, except that the norm of the error term is too large, as C_2 may be large. To contain the norm of this term, we recall the gadget matrix G from lecture 19. This will be used to contain the norm of C_2. We change the encryption algorithm once more:

- $\mathsf{Encrypt}(A, \mu \in \{0,1\})$: Sample $R \xleftarrow{R} \{0,1\}^{m \times m}$. Output $C = AR + \mu G$.

This meets a slightly different property: $s^T C = \mu s^T (AR + \mu G) = e^T R + \mu s^T G$. If the error term meets $\|e^T R\|_\infty \leq \frac{q}{4}$, we can successfully decrypt.

We will check for a multiplicative homomorphism now. Suppose we have ciphertexts $C_1 = AR_1 + \mu_1 G$ and $C_2 = AR_2 + \mu_2 G$. Then, the following holds:

$$\begin{aligned} C_1 G^{-1}(C_2) &= (AR_1 + \mu_1 G) G^{-1}(C_2) \\ &= AR_1 G^{-1}(C_2) + \mu_1 G G^{-1}(C_2) \\ &= AR_1 G^{-1}(C_2) + \mu_1 C_2 \\ &= AR_1 G^{-1}(C_2) + \mu_1 AR_2 + \mu_1 \mu_2 G \\ &= A(R_1 G^{-1}(C_2) + \mu_1 R_2) + \mu_1 \mu_2 G. \end{aligned}$$

Notice $G^{-1}(C_2) \in \{0,1\}^{m \times m}$. Then, the error $R_\times = R_1 G^{-1}(C_2) + \mu_1 R_2$ now meets

$$\|R_\times\|_\infty \leq m \|R_1\|_\infty + \|R_2\|_\infty.$$

This is the somewhat homomorphic encryption scheme we will use.

Analysis. Consider an initial ciphertext $C_1 = AR_1 + \mu_1 G$ with constant noise $\|R_1\|_\infty \leq 1$, and imagine repeatedly squaring it.

$$\begin{aligned} C_2 &= C_1 G^{-1}(C_1) && \text{noise } O(m) \\ &\vdots && \vdots \\ C_{d-1} &= C_{d-2} G^{-1}(C_{d-2}) && \text{noise } O(m^d). \end{aligned}$$

The GSW construction supports circuits of multiplicative depth d if $q > m^{O(d)}$.

Bootstrapping. In the next lecture, we will present a technique to "refresh" the randomness of a noisy ciphertext, allowing for the evaluation of larger circuits and getting us fully homomorphic encryption. Given a ciphertext C that encrypts μ with large noise, we will encrypt C and evaluate the decryption circuit homomorphically on this new ciphertext. This is known as Gentry's bootstrapping transformation.

Note that we can set the parameters in this construction. We just need to choose q large enough to support the decryption circuit in this case.

CS 395T: Topics in Cryptography April 4, 2024

Lecture 22: FHE Bootstrapping and Key Agreement from LWE

Lecturer: David Wu *Scribe: Matthew Healy*

From SWHE to FHE. In the previous lecture, we saw how to construct a somewhat homomorphic encryption (SWHE) scheme. The Gentry-Sahai-Waters (GSW) construction we introduced can support bounded depth computation where the LWE modulus is $q > m^{O(d)}$, with d as the multiplicative depth of the computation. In this lecture we will show how to obtain fully homomorphic encryption (FHE) from somewhat homomorphic encryption (SWHE) using Gentry's brilliant insight of bootstrapping.

High-level idea. Imagine we have a SWHE scheme that supports d operations. Suppose decryption in this scheme requires at most $(d-1)$ operations. Then, we can build an FHE scheme as follows:

The public key of the FHE scheme is the public key of the SWHE scheme (pk) and an encryption of the SWHE decryption key under the SWHE public key ($\mathsf{Encrypt}(\mathsf{pk}, \mathsf{sk})$). We now describe a ciphertext-refreshing procedure:

- For each SWHE ciphertext we associate a "noise" level that keeps track of how many more homomorphic operations can be performed on the ciphertext, while maintaining correctness. For example, we can perform d homomorphic operations on fresh ciphertexts, but after evaluating a single multiplication we can only perform $(d-1)$ more operations. The encryption of the SWHE decryption key under the SWHE public key has noise level 0.
- Suppose $\mathsf{ct}_\mu = \mathsf{Encrypt}(\mathsf{pk}, \mu)$. Using the SWHE scheme, we can compute $\mathsf{ct} = \mathsf{Encrypt}(\mathsf{pk}, f(\mu))$ with noise level d for any f computable using d homomorphic operations.
- We homomorphically compute $f_{\mathsf{ct}}(\mathsf{sk}) = \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct})$ which takes $d-1$ operations to compute.
- We homomorphically compute f_{ct} on $\mathsf{Encrypt}(\mathsf{pk}, \mathsf{sk}) \to \mathsf{Encrypt}(\mathsf{pk}, f_{\mathsf{ct}}(\mathsf{sk})) = \mathsf{Encrypt}(\mathsf{pk}, \mu)$ with noise level $(d-1)$.

After the refresh, the refreshed ciphertext can still support at least 1 more homomorphic operation since its current noise level is only $(d-1)$. By repeating this refresh whenever our ciphertexts become too noisy we can eventually perform unbounded depth computation. Thus, we have attained FHE.

Security. The security of this construction requires that this scheme should be secure even if the public key includes a copy of the decryption key. This requires us to make a "circular security" assumption. Obtaining FHE without circular security from LWE is currently an open problem.

Key Agreement from LWE. Analogous with the Diffie-Hellman key exchange protocol, we want to define a protocol for key exchange derived from LWE. We start with an amortized version of Regev's PKE scheme where each ciphertext encrypts a vector of bits. Since it may seem wasteful to use a vector to encrypt a single bit, we consider a simple variant where we reuse A to encrypt multiple bits.

Construction 22.1 (Encrypting multiple bits)**.** We describe this scheme as follows:

- $\mathsf{Setup}(1^\lambda, 1^l)$: Sample $A \xleftarrow{R} \mathbb{Z}_q^{n\times m}$, $S \xleftarrow{R} \mathbb{Z}_q^{n\times l}$, $E \leftarrow \chi^{m\times l}$. Output $\mathsf{pk} = (A, B^T = S^T A E^T \in \mathbb{Z}_q^{l\times m})$, $\mathsf{sk} = S$. One can think of S as l secret keys concatenated together.
- $\mathsf{Encrypt}(\mathsf{pk}, \mu \in \{0,1\}^l)$: Sample $r \xleftarrow{R} \{0,1\}^m$. Output $\mathsf{ct} = (Ar, B^T r + \mu \left\lfloor \frac{q}{2} \right\rfloor)$
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct})$: Output $\lfloor \mathsf{ct}_2 - S^T \mathsf{ct}_1 \rceil_{q,2}$.

Using this construction, we will define a protocol for key exchange relying on LWE which will operate as follows: Alice samples $A \xleftarrow{R} \mathbb{Z}_q^{n\times n}, S_1, E_1 \leftarrow \chi^{n\times k_1}, B_1^T = S_1^T A + E_1^T \in \mathbb{Z}_q^{k_1\times n}$ and sends A, B_1 to Bob. Bob samples $S_2, E_2 \leftarrow \chi^{n\times k_2}, B_2 \leftarrow AS_2 + E_2$ and sends B_2 to Alice. Alice then computes $\lceil S_1^T B_2 \rfloor_{2^l}$ while Bob computes $\lceil B_1^T S_2 \rfloor_{2^l}$. The computation within $\lceil \cdot \rfloor$ produces lk_1k_2 bits and we take the most significant l bits of the output as our shared key.

Correctness. To show correctness, observe that

$$S_1^T B_2 = S_1^T (AS_2 + E_2) = S_1^T AS_2 + S_1^T E_2$$

and

$$B_1^T S_2 = (S_1^T A + E_1^T) S_2 = S_1^T AS_2 + E_1^T S_2$$

We hope that $\lceil S_1^T B_2 \rfloor = \lceil B_1^T S_2 \rfloor$, which holds as long as $S_1^T B_2$ and $B_1^T S_2$ are far from a rounding boundary.

Security. We will not formally show the security of this scheme against passive eavesdroppers. At a high level, the distribution of the shared key is computationally close to uniform random even given the public messages, so Eve will only observe random matrices while Alice and Bob are performing the key exchange protocol.

CS 395T: Topics in Cryptography April 9, 2024

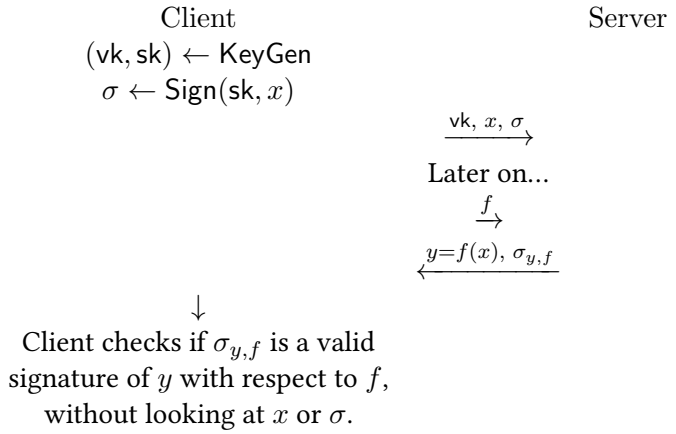
Lecture 23: Homomorphic Signatures

Lecturer: David Wu *Scribe: Eli Bradley*

In this lecture, we give a lattice-based construction of a homomorphic signature scheme [GPV08] and introduce key lattice equations from which many lattice-based constructions can be derived.

Recall lecture 14, in which we previously constructed a homomorphic signature scheme from a digital signature scheme, a CRHF and a SNARG for P. The construction we present today will be much lighter.

Homomorphic signatures. A homomorphic signature scheme is a digital signature scheme, with the additional property that arbitrary functions can be evaluated on the signed input. It would enable the following workflow:



We require the scheme meet the following requirements:

- Unforgeability: An efficient adversary given x cannot construct a signature on pair (y, f) where $y \neq f(x)$.
- Succinctness: For security parameter λ, $|\sigma_{y,f}| = \mathsf{poly}(\lambda.|y|)$.

Note that without the requirement of succinctness, there is a trivial solution: Simply output $\sigma_{y,f} = \sigma$, and let the verifier check $f(x) \stackrel{?}{=} y$ themselves. A similar observation can be made for the case of fully homomorphic encryption, where without a succinctness requirement, evaluation can be offloaded to the decrypter.

Upgrading succinctness. This succinctness requirement can be generically improved to remove dependence on $|y|$. Here are two ways this might be done:

- Define indicator function $f_y(x') = \begin{cases} 0 & f(x') \neq y \\ 1 & f(x') = y \end{cases}$. Homomorphically compute the signature σ_{1,f_y}.
- For a hash function H, define $g(x') = H(f(x'))$. Homomorphically compute the signature $\sigma_{H(m),g}$.

In the latter case, the function g is input-independent, which can be useful for precomputing.

Gentry-Peikert-Vaikuntanathan (GPV) construction [GPV08]. Our signature scheme will be the following. We assume without loss of generality the signing key sk contains vk.

- KeyGen: Sample $(A, T) \leftarrow \mathsf{TrapGen}(n, q)$ and $B_1, \ldots, B_l \xleftarrow{R} \mathbb{Z}_q^{n \times m}$.

  Output $\mathsf{vk} = (A, B_1, \ldots, B_l)$ and $\mathsf{sk} = T$.

- $\mathsf{Sign}(\mathsf{sk}, x \in \{0,1\}^l)$: For all $i \in [l]$, sample $R_i \leftarrow A^{-1}(B_i - x_i G)$. That is, use trapdoor T to sample short R_i such that $AR_i = B_i - x_i G$.

  Output $\sigma = (R_1, \ldots, R_l)$.

To verify the signature on bit $i \in [l]$, one would check $\|R_i\|_\infty$ is small and the verification relation $AR_i = B_i - x_i G$ holds.

Recall the equation for GSW ciphertexts: $B_i = AR_i + x_i G$, where x_i represented the message bit, R_i was the encryption randomness, and B_i was the ciphertext. This is merely a rearrangement of that same equation. This one equation or homomorphism, we will see, is the basis of most of lattice-based cryptography.

Homomorphic evaluation. We first consider the case of homomorphic evaluation of addition, $f(x) := x_1 + x_2$. By construction, we know $AR_1 = B_1 - x_1 G$ and $AR_2 = B_2 - x_2 G$. We can add these equations to get $A(R_1 + R_2) = (B_1 + B_2) - (x_1 + x_2)G$. Notice this has the form of the original verification relation. We define the signature $\sigma_{f(x),f}$ of $f(x)$ with respect to f, and the verification key vk_f, as follows.

$$A(\underbrace{R_1 + R_2}_{\sigma_{f(x),f}}) = (\underbrace{B_1 + B_2}_{\mathsf{vk}_f}) - (\underbrace{x_1 + x_2}_{f(x)})G$$

Now, consider the case of multiplication, $f(x) := x_1 x_2$. Here, we use the dual view, $B_1 = AR_1 + x_1 G$ and $B_2 = AR_2 + x_2 G$. We proceed just as in the GSW construction.

$$\begin{aligned} B_1 G^{-1}(B_2) &= (AR_1 + x_1 G) G^{-1}(B_2) \\ &= AR_1 G^{-1}(B_2) + x_1 B_2 \\ &= A(R_1 G^{-1}(B_2) + x_1 R_2) + x_1 x_2 G \\ A(\underbrace{R_1 G^{-1}(B_2) + x_1 R_2}_{\sigma_{f(x),f}}) &= \underbrace{B_1 G^{-1}(B_2)}_{\mathsf{vk}_f} - \underbrace{x_1 x_2}_{f(x)} G. \end{aligned}$$

Signatures and verification keys for arbitrary functions can be computed by combining these cases.

Example. As an example, we will compute the verification key for a circuit shown below.

TODO

This verification key vk_f is input-independent; it can be precomputed from vk and f without x.

Verification. We can now give the verification algorithm.

- $\mathsf{Verify}(\mathsf{vk}, y, f, \sigma)$: Map $(B_1, \ldots, B_l) \xmapsto{f} \mathsf{vk}_f = B_f$ as above.

  Check that $AR = B_f - yG$ and $\|R\|_\infty$ is small.

In this case, allowing R not to be short would break security, as opposed to breaking correctness in the GSW encryption scheme.

Signature evaluation. Observe the transformations applied to the signature are linear.

- Addition ($f(x) = x_i + x_j$): $R_+ = R_i + R_j = \begin{bmatrix} R_i & R_j \end{bmatrix} \begin{bmatrix} I_m \\ I_m \end{bmatrix}$.
- Multiplication ($f(x) = x_i x_j$): $R_\times = R_i G^{-1}(B_j) + x_i R_j = \begin{bmatrix} R_i & R_j \end{bmatrix} \begin{bmatrix} G^{-1}(B_j) \\ x_i I_m \end{bmatrix}$.

We assume Boolean circuits, so all wires have values $x_i \in \{0,1\}$. Then, the matrix multiplied on the right is $\{0,1\}$-valued in both cases. From this, we can conclude the following lattice homomorphism equation: For every function f computed a depth-d circuit and input x, there exists an evaluation map $H_{f,x}$ where $\|H_{f,x}\|_\infty \leq m^{O(d)}$ and

$$R_{f,x} = \begin{bmatrix} R_1 & \cdots & R_l \end{bmatrix} H_{f,x}.$$

Lattice homomorphism equations. Here, we will derive two key lattice equations from which both GSW scheme and the GPV scheme follow as corollaries.

Let $B = \begin{bmatrix} B_1 & \cdots & B_l \end{bmatrix}$. The verification key for f meets $B_f = B \cdot H_f$ for a short matrix H_f depending on f and B. We have shown the following verification relation.

$$\begin{aligned} AR_{f,x} &= B_f - f(x) \cdot G \\ A \begin{bmatrix} R_1 & \cdots & R_l \end{bmatrix} H_{f,x} &= \\ \begin{bmatrix} AR_1 & \cdots & AR_l \end{bmatrix} H_{f,x} &= \\ \begin{bmatrix} B_1 - x_1 G & \cdots & B_l - x_l G \end{bmatrix} H_{f,x} &= \\ (B - x^T \otimes G) H_{f,x} &= B_f - f(x) \cdot G. \end{aligned}$$

We get the following result.

Theorem 23.1. *Given any $B_1, \ldots, B_l \in \mathbb{Z}_q^{n \times m}$, any function f computed by a circuit of depth d, and any input x, there exists short matrices H_f and $H_{f,x}$ where the following hold.*

1. $B \cdot H_f = B_f$
2. $(B - x^T \otimes H_{f,x}) = B_f - f(x) \cdot G$

We can see the homomorphism of the GSW construction from this theorem. Recall the ciphertext was defined as $C = \begin{bmatrix} C_1 & \cdots & C_l \end{bmatrix}$ where $C_i = AR_i + x_i G$.

$$\begin{aligned} C - x^T \otimes G &= A \begin{bmatrix} R_1 & \cdots & R_l \end{bmatrix} && \text{(construction)} \\ (C - x^T \otimes G) H_{f,x} &= C_f - f(x) \cdot G && \text{(equation 2)} \\ &= C H_f - f(x) \cdot G && \text{(equation 1)} \\ C \cdot H_f &= (C - x^T \otimes G) H_{f,x} + f(x) \cdot G \\ &= A \underbrace{\begin{bmatrix} R_1 & \cdots & R_l \end{bmatrix} H_{f,x}}_{\text{encryption randomness}} + f(x) \cdot G. \end{aligned}$$

Thus, $C \cdot H_f$ is an encryption of $f(x)$.

Security. In the next lecture, we will argue the security of this homomorphic signature scheme, and discuss the dual notion of homomorphic commitments.

CS 395T: Topics in Cryptography April 11, 2024

Lecture 24: Homomorphic Commitments

Lecturer: David Wu *Scribe: Elahe Sadeghi*

Homomorphic Signatures. To recall the goal of a homomorphic signature schemes, it is to compute a signature σ_x to sign a message x, such that given (f, σ_x), one can evaluate $\sigma_{f,f(x)}$ that verifies with respect to the function f and the claim the value $f(x)$ (i.e. the verification algorithm does not need to know x). It is useful in the settings where the input x is sensitive. One application of this can be when someone outsources a sensitive dataset, and other people can query the dataset, hoping to learn the output of the query, while the dataset is hidden. The construction was introduces by adapting the GSW homomorphic encryption scheme.

Construction 24.1 (Homomorphic Signature)**.** The homomorphic signature scheme $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Verify})$ adapting the GSW homomorphic encryption scheme has been constructed as follows:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$: The setup algorithm takes a security parameter λ as input, and then obtains $(\mathbf{A}, \mathbf{T}) \xleftarrow{R} \mathsf{TrapGen}(n, q)$ and samples ℓ random $n \times m$ matrices $\mathbf{B}_1, \ldots, \mathbf{B}_\ell \xleftarrow{R} \mathbb{Z}_q^{n \times m}$. It then sets $\mathsf{vk} \coloneqq (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_\ell)$ and $\mathsf{sk} \coloneqq \mathbf{T}$.
- $\mathsf{Sign}(\mathsf{sk}, \mathbf{x}) \to \sigma$: The signing algorithm takes as inputs a signing key $\mathsf{sk} = \mathbf{T}$ and a message $\mathbf{x} \in \{0,1\}^\ell$ as inputs. It then uses the trapdoor $\mathbf{T}$ to compute targeted pre-images $\mathbf{R}_i$ such that $\mathbf{A}\mathbf{R}_i = \mathbf{B}_i - x_i\mathbf{G}$ (i.e. $\mathbf{R}_i \leftarrow \mathbf{A}^{-1}(\mathbf{B}_i - x_i\mathbf{G})$), and sets $\sigma \coloneqq (\mathbf{R}_1, \ldots, \mathbf{R}_\ell)$.
- $\mathsf{Eval}(\mathsf{vk}, \sigma_\mathbf{x}, f) \to \sigma_{f,f(\mathbf{x})}$: The evaluation algorithm takes a verification key $\mathsf{vk} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_\ell)$, a signature $\sigma_\mathbf{x} = (\mathbf{R}_1, \ldots, \mathbf{R}_\ell)$, and a function f as inputs, and it computes $\sigma_{f,f(\mathbf{x})} \coloneqq \mathbf{R} \cdot \mathbf{H}_{f,\mathbf{x}}$, where $\mathbf{R} = [\mathbf{R}_1 \mid \cdots \mid \mathbf{R}_\ell]$ and $\mathbf{H}_{f,\mathbf{x}}$ is the evaluation matrix.
- $\mathsf{Verify}(\mathsf{vk}, f, y, \sigma_{f,f(\mathbf{x})}) \to 0/1$: The verification algorithm takes a verification key $\mathsf{vk} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_\ell)$, a function f, a value y, and a signature $\sigma_{f,f(\mathbf{x})} = (\mathbf{R}_1, \ldots, \mathbf{R}_\ell)$ as inputs. If $||\sigma_{f,f(\mathbf{x})}||_\infty$ is small and $\mathbf{A}\sigma_{f,f(\mathbf{x})} = \mathbf{B}_f - y \cdot \mathbf{G}$, the algorithm outputs 1, and 0 otherwise.

Homomorphism or homomorphic evaluations in signature schemes relies on matrix evaluation. From the signing algorithm, we have the key equation $\mathbf{B} = \mathbf{A}\mathbf{R} + \mathbf{x} \otimes \mathbf{G}$, where $\mathbf{B} = [\mathbf{B}_1 \mid \cdots \mid \mathbf{B}_\ell]$ and $\mathbf{R} = [\mathbf{R}_1 \mid \cdots \mid \mathbf{R}_\ell]$. Observe that $\mathbf{x} \otimes \mathbf{G} = [x_1\mathbf{G} \mid \cdots \mid x_\ell\mathbf{G}]$.
Let $\mathbf{B}_1, \ldots, \mathbf{B}_\ell \in \mathbb{Z}_q^{n \times m}$ be arbitrary matrices. We can define two homomorphic evaluation algorithms as follows.

1. For every function $f : \{0,1\}^\ell \to \{0,1\}$, there exists a short matrix $\mathbf{H}_f$ such that
$$\mathbf{B} \cdot \mathbf{H}_f = \mathbf{B}_f$$
2. For every $f : \{0,1\}^\ell \to \{0,1\}$ and every $\mathbf{x} \in \{0,1\}^\ell$, there exists a short matrix $\mathbf{H}_{f,\mathbf{x}}$ such that
$$(\mathbf{B} - \mathbf{x} \otimes \mathbf{G}) \cdot \mathbf{H}_{f,\mathbf{x}} = \mathbf{B}_f - f(\mathbf{x}) \cdot \mathbf{G}$$
Intuitively, $(\mathbf{B} - \mathbf{x} \otimes \mathbf{G})$ is an encoding of $\mathbf{x}$ with respect to $\mathbf{B}$, in which applying the evaluation matrix $\mathbf{H}_{f,\mathbf{x}}$ on it computes a new encoding $\mathbf{B}_f - f(\mathbf{x}) \cdot \mathbf{G}$ of $f(\mathbf{x})$ with respect to $\mathbf{B}_f$, where $\mathbf{B}_f$ is a function depending only on public parameters.

Correctness:

$$\mathbf{A} \cdot \sigma_{f,f(\mathbf{x})} = \mathbf{ARH}_{f,\mathbf{x}} = (\mathbf{B} - \mathbf{x} \otimes \mathbf{G}) \cdot \mathbf{H}_{f,\mathbf{x}} = \mathbf{B}_f - f(\mathbf{x}) \cdot \mathbf{G} = \mathbf{B}_f - y \cdot \mathbf{G}$$

Since $\mathbf{R}$ is small (as it is sampled from the Gaussian distribution), and $\mathbf{H}_{f,\mathbf{x}}$ is also small (and as said in previous lectures, $||\mathbf{H}_f||_\infty, ||\mathbf{H}_{f,\mathbf{x}}||_\infty \leq m^{O(d)}$ where d is the depth of the circuit that computes f). Thus, $\sigma_{f,f(\mathbf{x})} = \mathbf{R} \cdot \mathbf{H}_{f,\mathbf{x}}$ is going to be small. □

Security. We show that the above construction has *selective security*. Selective security is defined as follows, and can be boosted to *adaptive* security *without* additional assumptions.

1. The adversary $\mathcal{A}$ sends $\mathbf{x} \in \{0,1\}^\ell$ to the selective security challenger.
2. The challenger computes and sends vk and $\sigma_{\mathbf{x}}$ to the adversary $\mathcal{A}$.
3. The adversary $\mathcal{A}$ sends $(f, y, \sigma_{f,y})$ to the challenger, and wins if $\mathsf{Verify}(\mathsf{vk}, f, y, \sigma_{f,y}) = 1$ and $y \neq f(\mathbf{x})$.

Claim 24.2. *The above construction Construction 24.1 is selective secure under* SIS *assumption.*

Proof. A sketch of the reduction would be as follows:
Assume that $\mathcal{A}$ is an adversary breaking the selective security of the signature scheme. Construct $\mathcal{B}$ breaking the SIS as follows:

1. $\mathcal{B}$ receives the challenge matrix $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$ from its SIS challenger, and $\mathbf{x} \in \{0,1\}^\ell$ from the adversary $\mathcal{A}$.
2. $\mathcal{B}$ then samples $\mathbf{R}_1, \ldots, \mathbf{R}_\ell \xleftarrow{R} \{0,1\}^{m \times m}$, and computes $\mathbf{B}_i \leftarrow \mathbf{AR}_i + x_i \mathbf{G}$ for all $i \in [\ell]$. Then, $\mathcal{B}$ sends $\mathsf{vk} \coloneqq (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_\ell)$ and $\sigma_{\mathbf{x}} \coloneqq (\mathbf{R}_1, \ldots, \mathbf{R}_\ell)$ to $\mathcal{A}$. Note that by the LHL, the distribution of $\mathbf{B}_i$'s are uniform, as $\mathbf{A}$ and $\mathbf{R}_i$'s are random. Therefore, the distributions match the real scheme.
3. $\mathcal{B}$ receives $(f, y, \sigma_{f,y})$ from $\mathcal{A}$, and computes and sends $(\mathbf{RH}_{f,\mathbf{x}} - \mathbf{R}_{f,y})\, \mathbf{G}^{-1}(0)$ to its challenger.

If $\mathcal{A}$ succeeds with non-negligible probability ϵ, then we can show that $(\mathbf{RH}_{f,\mathbf{x}} - \mathbf{R}_{f,y})\, \mathbf{G}^{-1}(0)$ is a solution to SIS problem with respect to $\mathbf{A}$. First, observe that by LHL, $\mathbf{B}_i$'s are uniform (and of the right distribution of the signature scheme). Thus, upon receiving $(f, y \in \{0,1\}, \sigma_{f,y} = \mathbf{R}_{f,y})$, and assuming $\mathcal{A}$ succeeds, one can conclude that $||\mathbf{R}_{f,y}||_\infty$ is small and $\mathbf{AR}_{f,y} = \mathbf{B}_f - y \cdot \mathbf{G}$, while $y \neq f(\mathbf{x})$.
Based on the correctness of the signature scheme, $\mathbf{ARH}_{f,\mathbf{x}} = \mathbf{B}_f - f(\mathbf{x}) \cdot \mathbf{G}$ where $\mathbf{R} = [\mathbf{R}_1 \mid \cdots \mid \mathbf{R}_\ell]$. From $\mathcal{A}$, $\mathbf{AR}_{f,y} = \mathbf{B}_f - y \cdot \mathbf{G}$. Thus,

$$\mathbf{A}\,(\mathbf{RH}_{f,\mathbf{x}} - \mathbf{R}_{f,y}) = (y - f(\mathbf{x})) \cdot \mathbf{G}$$

Note that $y - f(\mathbf{x}) \neq 0$ implying that $y - f(\mathbf{x}) \in \{-1, +1\}$. Therefore,

$$\mathbf{A}\,(\mathbf{RH}_{f,\mathbf{x}} - \mathbf{R}_{f,y}) = \pm \mathbf{G}$$

implying that $(\mathbf{RH}_{f,\mathbf{x}} - \mathbf{R}_{f,y})$ is the trapdoor w.r.t. $\mathbf{A}$. Having the above equation, $\mathcal{B}$ can find the SIS solution w.r.t. $\mathbf{A}$ by computing $\mathbf{G}^{-1}(0)$ having the trapdoor.

$$\mathbf{A}\,(\mathbf{RH}_{f,\mathbf{x}} - \mathbf{R}_{f,y})\, \mathbf{G}^{-1}(0) = 0$$

So $(\mathbf{RH}_{f,\mathbf{x}} - \mathbf{R}_{f,y})\, \mathbf{G}^{-1}(0)$ is going to be the SIS solution. Since the pre-image sampling is outputting a random vector, this value will be non-zero with high probability. □ □

Privacy. In applications, there are times that we might want the signature $\sigma_{f,f(\mathbf{x})} = \mathbf{R}\mathbf{H}_{f,\mathbf{x}}$ to hide information about $\mathbf{x}$. This is not the case in the current scheme as the evaluation matrix $\mathbf{H}_{f,\mathbf{x}}$ depends on $\mathbf{x}$, and in fact, is a function of $\mathbf{x}$. Observe that $\mathbf{R}_i$'s have Gaussian distributions, which multiplying it with a matrix that depends on $\mathbf{x}$, will leak information about $\mathbf{x}$ by looking at the covariance of the vectors that is going to depend on $\mathbf{H}_{f,\mathbf{x}}$. So the question is

Is there a way to make the signature private?

In previous lectures, we faced a similar problem in our signature scheme, where the signature $\mathbf{R} \cdot \mathbf{G}^{-1}(H(m))$, was deterministic. To solve this, we used $\mathbf{R}$ as input to the pre-image sampler (as a trapdoor) to make the signature randomized (so that the signature does not depend on the trapdoor). We will take a similar approach for randomization here:
From the signing algorithm, $\mathbf{A}\mathbf{R}_{f,f(\mathbf{x})} = \mathbf{B}_f - f(\mathbf{x})\mathbf{G}$, where $\mathbf{R}_{f,f(\mathbf{x})} = \mathbf{R} \cdot \mathbf{H}_{f,\mathbf{x}}$ is the signature on $f(\mathbf{x})$. We can rewrite this as $\mathbf{A}\mathbf{R}_{f,f(\mathbf{x})} = \mathbf{B}_f - y \cdot \mathbf{G}$, where $y = f(\mathbf{x}) \in \{0,1\}$.
We change our verification relation as follows: define a verification matrix $\mathbf{V} = [\mathbf{A} \mid \mathbf{B}_f + (y-1) \cdot \mathbf{G}]$. Rewrite $\mathbf{V}$ as follows:

$$\begin{aligned}\mathbf{V} &= [\mathbf{A} \mid \mathbf{B}_f + (y-1) \cdot \mathbf{G}] \\ &= \left[\mathbf{A} \mid \mathbf{A}\mathbf{R}_{f,f(\mathbf{x})} + (2y-1) \cdot \mathbf{G}\right] \\ &= \left[\mathbf{A} \mid \mathbf{A}\mathbf{R}_{f,f(\mathbf{x})} + (\pm\mathbf{G})\right]\end{aligned}$$

Thus, we can find a trapdoor for this matrix as follows:

$$\mathbf{V} = \left[\mathbf{A} \mid \mathbf{A}\mathbf{R}_{f,f(\mathbf{x})} + (\pm\mathbf{G})\right] \cdot \begin{bmatrix} -\mathbf{R}_{f,f(\mathbf{x})} \\ \mathbf{I} \end{bmatrix} = \pm\mathbf{G}$$

implying that $\begin{bmatrix} -\mathbf{R}_{f,f(\mathbf{x})} \\ \mathbf{I} \end{bmatrix}$ is a trapdoor for $\mathbf{V}$.
Therefore, to re-randomize the solution, we just need to find a fresh solution from this lattice. So we are going to prove that we have a signature without revealing the trapdoor, and we use this to as a pre-image sampler. So we say that we have a trapdoor for this lattice because we have a solution for the SIS problem. This property is called "context-hiding". In particular, we show that we can find a short $\mathbf{t}$ such that $\mathbf{V}\mathbf{t} = 0$, which hides the trapdoor, and can be used to randomize. This gives us needed randomization for the signature scheme, to hide the original input, and achieve privacy. □

Construction 24.3 (Homomorphic Commitments)**.** These primitives can be viewed as dual of "homomorphic signatures". The idea behind homomorphic commitments is to commit to a value $\mathbf{x}$, and open to $f(\mathbf{x})$. A vanilla idea of the scheme is sketched in the following:

- Public parameters pp: $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$.
- Commit to $\mathbf{x}$: Sample $\mathbf{R}_i \xleftarrow{R} \{0,1\}^{m \times m}$, and compute $\mathbf{C}_i \leftarrow \mathbf{A}\mathbf{R}_i + x_i\mathbf{G}$ for all $i \in [\ell]$. Let $\mathbf{C} = [\mathbf{C}_1 \mid \cdots \mid \mathbf{C}_\ell]$ and $\mathbf{R} = [\mathbf{R}_1 \mid \cdots \mid \mathbf{R}_\ell]$, and thus, $\mathbf{C} = \mathbf{A}\mathbf{R} + \mathbf{x} \otimes \mathbf{G}$.
- Open to $f(\mathbf{x})$: Compute $\mathbf{R}_{f,f(\mathbf{x})} \coloneqq \mathbf{R} \cdot \mathbf{H}_{f,\mathbf{x}}$
- Verification relation: Check the equality $\mathbf{A}\mathbf{R}_{f,f(\mathbf{x})} + y\mathbf{G} = \mathbf{C}_f$

Correctness: If $y = f(\mathbf{x})$, then

$$\mathbf{ARH}_{f,f(\mathbf{x})} = (\mathbf{C} - \mathbf{x} \otimes \mathbf{G})\,\mathbf{H}_{f,\mathbf{x}} = \mathbf{C}_f - f(\mathbf{x})\mathbf{G}$$

Observe that the commitments in this scheme can be really large. In particular, if we commit to a ℓ-bit string, the commitment contains of ℓ matrices. So we want to construct *succinct commitments* (aka "functional commitments").
The question if

How can we compress?

Idea: Suppose pp contains the extra information as follows:

$$\left[\begin{array}{ccc|c} \mathbf{A} & & & \mathbf{W}_1 \\ & \mathbf{A} & & \mathbf{W}_2 \\ & & \ddots & \vdots \\ & & \mathbf{A} & \mathbf{W}_\ell \end{array}\right] (\mathbf{G})$$

where $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m}$ and $\mathbf{W}_i \xleftarrow{R} \mathbb{Z}_q^{n\times m}$ for all $i \in [\ell]$. For simplicity, define $\mathbf{B}_\ell$ as

$$\mathbf{B}_\ell = \left[\begin{array}{ccc|c} \mathbf{A} & & & \mathbf{W}_1 \\ & \mathbf{A} & & \mathbf{W}_2 \\ & & \ddots & \vdots \\ & & \mathbf{A} & \mathbf{W}_\ell \end{array}\right]$$

Definition 24.4 (ℓ-succinct SIS)**.** SIS is hard even with respect to $\mathbf{A}$ given $\mathbf{B}_\ell^{-1}(\mathbf{G})$.

Note that $\mathbf{B}_\ell^{-1}(\mathbf{G})$ is a trapdoor for $\mathbf{B}_\ell$:

$$\left[\begin{array}{ccc|c} \mathbf{A} & & & \mathbf{W}_1 \\ & \mathbf{A} & & \mathbf{W}_2 \\ & & \ddots & \vdots \\ & & \mathbf{A} & \mathbf{W}_\ell \end{array}\right] \cdot \begin{bmatrix} \mathbf{R}_1 \\ \vdots \\ \mathbf{R}_\ell \\ \mathbf{T} \end{bmatrix} = 0$$

which gives us the following linear system

$$\begin{cases} \mathbf{AR}_1 + \mathbf{W}_1\mathbf{T} = 0 \\ \vdots \\ \mathbf{AR}_\ell + \mathbf{W}_\ell\mathbf{T} = 0 \end{cases}$$

As $\mathbf{W}_i\mathbf{T}$ has a uniform distribution, it is going to be hard to find a SIS solution w.r.t. $\mathbf{A}$ from this system. In next lecture, we see how to use the structure of this trapdoor (i.e. this ℓ-succinct SIS problem) as a way to get succinct homomorphic commitments, known as "functional commitments", which gives us functional commitments for all circuits.

CS 395T: Topics in Cryptography April 16, 2024

Lecture 25: Functional Commitments

Lecturer: David Wu *Scribe: Elahe Sadeghi*

Recap (Lattice Homomorphism). Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m\ell}$, where $m = \Omega(n \log(q))$.

1. For all functions $f : \{0,1\}^\ell \to \{0,1\}$ computable by circuit of depth d, we can construct a matrix $\mathbf{H}_f \in \mathbb{Z}_q^{m\ell \times m}$, where $||\mathbf{H}_f||_\infty \leq m^{O(d)}$.
2. For all functions $f : \{0,1\}^\ell \to \{0,1\}$ and all $\mathbf{x} \in \{0,1\}^\ell$, we can construct a matrix $\mathbf{H}_{f,\mathbf{x}} \in \mathbb{Z}_q^{m\ell \times m}$, where $||\mathbf{H}_{f,\mathbf{x}}||_\infty \leq m^{O(d)}$.

From these matrices, we get "key equation" that is used in lattice-based cryptography:

$$\left(\mathbf{A} - \left(\mathbf{x}^\mathsf{T} \otimes \mathbf{G}\right)\right) \cdot \mathbf{H}_{f,\mathbf{x}} = \mathbf{A} \cdot \mathbf{H}_f - f(\mathbf{x}) \cdot \mathbf{G}$$

where we denote $\mathbf{A}_f \coloneqq \mathbf{A} \cdot \mathbf{H}_f$ and $\left(\mathbf{A} - \left(\mathbf{x}^\mathsf{T} \otimes \mathbf{G}\right)\right) = [\mathbf{A}_1 - x_1\mathbf{G} \mid \cdots \mid \mathbf{A}_\ell - x_\ell\mathbf{G}]$. Intuitively, the left hand-side of the above equation is multiplying an "encoding of $\mathbf{x}$ w.r.t. $\mathbf{A}$" by an evaluation matrix $\mathbf{H}_{f,\mathbf{x}}$, and derive an "encoding of $f(\mathbf{x})$ w.r.t. $\mathbf{A}_f$" on the right hand-side.

Functional Commitments. One of the motivations/applications of these commitments is the scenario that a bank wants to design a "single" fair algorithm to apply it to every client requesting a loan. Then the bank can commit to the algorithm to decide whether one is eligible for getting loans, take some input (client's loan application), and then prove to client that the output value is consistent with the algorithm. These primitives have two dual purposes of 1. commit to $\mathbf{x}$ and open to evaluation $f(\mathbf{x})$, and 2. commit to f and open to a value at $f(\mathbf{x})$. These two are equivalent and in fact, from 1, one can define a universal circuit as $U_\mathbf{x}(f) \coloneqq f(\mathbf{x})$, to derive 2.
The goal of this primitive is to have *succinctness* in commitments and openings. (i.e. they both are sub-linear in the length of the input).
In the previous lecture, we defined a construction of homomorphic commitments where the commitments were large. The goal is to construct a functional commitment derived from the previous construction with a compression technique on top! To recall, Construction 24.3 was defined as follows:

- Public parameters pp: Sample $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$.
- Commit to $\mathbf{x} \in \{0,1\}^\ell$: Sample $\mathbf{R} \xleftarrow{R} \{0,1\}^{m \times m\ell}$. Compute $\mathbf{C} = \mathbf{A}\mathbf{R} + \left(\mathbf{x}^\mathsf{T} \otimes \mathbf{G}\right)$, where $\mathbf{C}$ is the commitment and $\mathbf{R} = [\mathbf{R}_1 \mid \cdots \mid \mathbf{R}_\ell]$ is the opening.
- Opening to function $f : \{0,1\}^\ell \to \{0,1\}$: Compute $\left(\mathbf{C} - \mathbf{x}^\mathsf{T} \otimes \mathbf{G}\right) \mathbf{H}_{f,\mathbf{x}} = \mathbf{C}\mathbf{H}_f - f(\mathbf{x})\mathbf{G}$, where from above, $\left(\mathbf{C} - \mathbf{x}^\mathsf{T} \otimes \mathbf{G}\right) = \mathbf{A}\mathbf{R}$, which gives us the following:

  $$\mathbf{A}\left(\mathbf{R}\mathbf{H}_{f,\mathbf{x}}\right) + f(\mathbf{x})\mathbf{G} = \mathbf{C}\mathbf{H}_f$$

  Denote $\mathbf{R}_f \coloneqq \mathbf{R}\mathbf{H}_{f,\mathbf{x}} \in \mathbb{Z}_q^{m \times m}$ and $\mathbf{C}_f \coloneqq \mathbf{C}\mathbf{H}_f$, implying that $\mathbf{C}_f$ is the new commitment and $\mathbf{R}_f$ is the new opening.

- Verification relation: Given $(\mathsf{pp}, \mathbf{C}, f, y, \mathbf{R}_f)$, the verifier computes $\mathbf{H}_f$ as it only depends on f and is computable, and then computes $\mathbf{C}_f = \mathbf{C}\mathbf{H}_f$. It then checks if $\mathbf{C}_f = \mathbf{A}\mathbf{R}_f + y\mathbf{G}$, and whether $||\mathbf{R}_f||_\infty$ is small.

Observe that the openings in this scheme are in $\mathbb{Z}_q^{m\times m}$, which considered to be sub-linear in the length of the input, ℓ, while the commitments are in $\mathbb{Z}_q^{n\times m\ell}$, which is large.

Definition 25.1 (ℓ-succinct SIS)**.** Suppose $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m}$ and $\mathbf{W}_1, \ldots, \mathbf{W}_\ell \xleftarrow{R} \mathbb{Z}_q^{n\times t}$. Let

$$\mathbf{B} = \left[\begin{array}{cccc|c} \mathbf{A} & & & & \mathbf{W}_1 \\ & \mathbf{A} & & & \mathbf{W}_2 \\ & & \ddots & & \vdots \\ & & & \mathbf{A} & \mathbf{W}_\ell \end{array}\right]$$

ℓ-succinct SIS problem (aka "SIS with hints") states that given $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{B}^{-1}(\mathbf{G})$ (a trapdoor for $\mathbf{B}$), it is hard to find a non-zero solution $\mathbf{x} \in \mathbb{Z}_q^m$, such that $\mathbf{A}\mathbf{x} = 0 \mod q$, and $||\mathbf{x}||_\infty \leq \beta$.
The choice of parameter t (the dimension of the hint matrices), is important to ensure that the hints will not help with breaking the SIS. So if $t > m\ell$, then SIS implies ℓ-succinct SIS.

Observe that the hints in the above assumption is not going to immediately help to break the SIS assumption. Suppose $\mathbf{B}^{-1}(0)$ is given as

$$\mathbf{B}^{-1}(0) = \begin{bmatrix} \mathbf{T}_1 \\ \vdots \\ \mathbf{T}_\ell \\ \mathbf{U} \end{bmatrix} \Rightarrow \begin{cases} \mathbf{A}\mathbf{T}_1 + \mathbf{W}_1\mathbf{U} = 0 \\ \vdots \\ \mathbf{A}\mathbf{T}_\ell + \mathbf{W}_\ell\mathbf{U} = 0 \end{cases}$$

which is not going to help with finding the SIS solution immediately as $\mathbf{W}_i\mathbf{U}$'s distribution is uniform.

Homomorphic Commitment Compression. The idea is to use a variant of ℓ-succinct SIS assumption in which $t = m$, in order to compress the commitments in Construction 24.3. In that construction, a commitment was represented as $\overline{\mathbf{C}} = [\mathbf{C}_1 \mid \cdots \mid \mathbf{C}_\ell]$, where

$$\begin{cases} \mathbf{C}_1 = \mathbf{A}\mathbf{R}_1 + x_1\mathbf{G} \\ \vdots \\ \mathbf{C}_\ell = \mathbf{A}\mathbf{R}_\ell + x_\ell\mathbf{G} \end{cases}$$

The distribution of $\overline{\mathbf{C}}$ was said to be uniform by LHL, as the matrices $\mathbf{R}_i$'s are chosen randomly. However, compressing randomness, without losing information, is hard, and therefore, the idea is to define the commitments $\mathbf{C}_i$'s correlated.
Suppose $\mathbf{C}_i = \mathbf{W}_i\mathbf{C}$, for $\mathbf{C} \in \mathbb{Z}_q^{m\times m}$. In this case, the commitment algorithm only needs to output $\mathbf{C}$, as all the commitments $\mathbf{C}_1, \ldots, \mathbf{C}_\ell$ are computable assuming $\mathbf{W}_i$'s are known publicly. Thus, the goal is to compute such a $\mathbf{C}$.
$\mathbf{C}$ should sastisfy the following system:

$$\begin{cases} \mathbf{W}_1\mathbf{C} = \mathbf{A}\mathbf{R}_1 + x_1\mathbf{G} \\ \vdots \\ \mathbf{W}_\ell\mathbf{C} = \mathbf{A}\mathbf{R}_\ell + x_\ell\mathbf{G} \end{cases} \Rightarrow \left[\begin{array}{cccc|c} \mathbf{A} & & & & \mathbf{W}_1 \\ & \mathbf{A} & & & \mathbf{W}_2 \\ & & \ddots & & \vdots \\ & & & \mathbf{A} & \mathbf{W}_\ell \end{array}\right] \cdot \begin{bmatrix} \mathbf{R}_1 \\ \vdots \\ \mathbf{R}_\ell \\ -\mathbf{C} \end{bmatrix} = \begin{bmatrix} -x_1\mathbf{G} \\ \vdots \\ -x_\ell\mathbf{G} \end{bmatrix}$$

Based on ℓ-succinct SIS assumption, $\mathbf{B} = \left[\begin{array}{ccc|c} \mathbf{A} & & & \mathbf{W}_1 \\ & \mathbf{A} & & \mathbf{W}_2 \\ & & \ddots & \vdots \\ & & \mathbf{A} & \mathbf{W}_\ell \end{array}\right]$. Therefore, it suffices to publish the trapdoor $\mathbf{B}^{-1}(\mathbf{G})$ in the public parameters pp.

Construction 25.2 (Functional Commitment)**.** We can construct a functional commitment scheme which is similar to Construction 24.3 with the following changes:

- Public parameter pp: $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\mathbf{W}_1, \ldots, \mathbf{W}_\ell$, and $\mathbf{B}^{-1}(\mathbf{G})$
- Commit to $\mathbf{x}$: Compute $\mathbf{C}, \mathbf{R}_1, \ldots, \mathbf{R}_\ell$ by using the trapdoor $\mathbf{B}^{-1}(\mathbf{G})$.

Security. The security experiment between the challenger and the adversary $\mathcal{A}$ as follows:

1. The challenger sends $\mathbf{A}, \mathbf{W}_1, \ldots, \mathbf{W}_\ell, \mathbf{B}^{-1}(\mathbf{G})$ to the adversary.
2. The adversary will eventually send $(\mathbf{C}, f, \mathbf{R}_0, \mathbf{R}_1)$, and for $\begin{cases} \mathbf{C}_1 = \mathbf{W}_1\mathbf{C} \\ \vdots \\ \mathbf{C} - \ell = \mathbf{W}_\ell\mathbf{C} \end{cases}$ and $\mathbf{C}_f = [\mathbf{C}_1 \mid \cdots \mid \mathbf{C}_\ell]$, the adversary wins if $\begin{cases} \mathbf{C}_f = \mathbf{A}\mathbf{R}_0 \\ \mathbf{C}_f = \mathbf{A}\mathbf{R}_1 + \mathbf{G} \end{cases}$ and $\mathbf{R}_0, \mathbf{R}_1$ are small.

We say a functional commitment scheme is secure if the advantage of any efficient adversary in above experiment is negligible.

Claim 25.3. *Construction 25.2 is secure under ℓ-succinct* SIS *assumption.*

Proof. sketch: If there exists an adversary $\mathcal{A}$ against the security experiment of Construction 25.2 with non-negligible advantage, then we can construct the following adversary $\mathcal{B}$ solving ℓ-succinct SIS with non-negligible advantage (as $\mathcal{A}$'s advantage):

1. $\mathcal{B}$ receives $\mathbf{A}, \mathbf{W}_1, \ldots, \mathbf{W}_\ell, \mathbf{B}^{-1}(\mathbf{G})$ from its challenger and forwards it to a$\mathcal{A}$.
2. It receives $(\mathbf{C}, f, \mathbf{R}_0, \mathbf{R}_1)$ from $\mathcal{A}$ where $\begin{cases} \mathbf{C}_f = \mathbf{A}\mathbf{R}_0 \\ \mathbf{C}_f = \mathbf{A}\mathbf{R}_1 + \mathbf{G} \end{cases}$ and $\mathbf{R}_0, \mathbf{R}_1$ are small.
3. $\mathcal{B}$ computes $\mathbf{A}(\mathbf{R}_0 - \mathbf{R}_1) = \mathbf{G}$, and therefore a trapdoor with respect to $\mathbf{A}$. $\mathcal{B}$ then uses the trapdoor to solve the SIS assumption.

□

Intro to Attribute-Based Encryption for circuits. To recall the intuitive notion of ABEs, the scheme contains the following algorithms:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{msk}, \mathsf{mpk})$
- $\mathsf{Encrypt}(\mathsf{mpk}, x, m) \to \mathsf{ct}$

- $\mathsf{KeyGen}(\mathsf{msk}, f) \to \mathsf{sk}_f$
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to m$: If $f(x) = 0$, then the decryption algorithm is able to decrypt, and not otherwise.

From LWE assumption, we can construct an ABE scheme that supports arbitrary policies for the keys, and support any circuit f. To start, we use the idea of the "dual Regev encryption scheme". The dual Regev encryption scheme is as follows:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$: Given the security parameter λ, the key generation algorithm samples $\mathbf{A} \xleftarrow{R} \mathbb{Z}_1^{n \times m}$ and $\mathbf{r} \xleftarrow{R} \{0,1\}^m$, and sets $\mathsf{pk} := (\mathbf{A}, \mathbf{b} = \mathbf{Ar})$ and $\mathsf{sk} := \mathbf{r}$.
- $\mathsf{Encrypt}(\mathsf{pk}, \mu \in \{0,1\}) \to \mathsf{ct}$: Given the public key pk and the message μ, the encryption algorithm first samples $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \xleftarrow{R} \chi^m$, and $e' \xleftarrow{R} \chi$. It then outputs $\mathsf{ct} := (\mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T}, \mathbf{s}^\mathsf{T}\mathbf{b} + e' + \mu\lfloor \frac{q}{2} \rceil)$.
- $\mathsf{Decrypt}(\mathsf{sk}, (\mathsf{ct}_1, \mathsf{ct}_2)) \to \mu$: Given the secret key sk and the ciphertext $(\mathsf{ct}_1, \mathsf{ct}_2)$, the decryption algorithm computes $\mathsf{ct}_2 - \mathsf{ct}_1^\mathsf{T}\mathbf{r} = \mu\lfloor \frac{q}{2} \rceil + e' + \mathbf{e}^\mathsf{T}\mathbf{r}$.

A comparison of the "dual Regev" and "Primal Regev" encryption schemes is illustrated in Fig. 25.1.
The security of this scheme is based on LHL and LWE. Based on the leftover hash lemma, $\mathbf{b} = \mathbf{Ar}$ is uniformly random, and therefore, by LWE, we can show that μ gets blinded by a uniform string.
Observe that $\mathbf{r}$ is a special component in dual Regev construction as having and LWE instance $(\mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T})$ with respect to $\mathbf{A}$, we get a new LWE instance by multiplying with $\mathbf{r}$, as $(\mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T})\mathbf{r} = \mathbf{s}^\mathsf{T}\mathbf{b} + \mathbf{e}^\mathsf{T}\mathbf{r}$, with respect to $\mathbf{b}$.

Dual Regev	**Primal Regev**
• KeyGen: $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\mathbf{r} \xleftarrow{R} \{0,1\}^m$. $\mathsf{pk} := (\mathbf{A}, \mathbf{b} = \mathbf{Ar})$ and $\mathsf{sk} := \mathbf{r}$. • Encrypt: $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \xleftarrow{R} \chi^m$ and $e' \xleftarrow{R} \chi$. $\mathsf{ct} := (\mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T}, \mathbf{s}^\mathsf{T}\mathbf{b} + e' + \mu\lfloor \frac{q}{2} \rceil)$	• KeyGen: $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, and $\mathbf{e} \xleftarrow{R} \chi^m$. $\mathsf{pk} := (\mathbf{A}, \mathbf{b} = \mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T})$ and $\mathsf{sk} := \mathbf{s}$. • Encrypt: $\mathbf{r} \xleftarrow{R} \{0,1\}^m$. $\mathsf{ct} := (\mathbf{Ar}, \mathbf{b}\mathsf{T} + \mu\lfloor \frac{q}{2} \rceil)$

Figure 25.1: Comparison of Dual Regev and Primal Regev Encryption Schemes.

CS 395T: Topics in Cryptography April 18, 2024

Lecture 26: Attribute Based Encryption

Lecturer: David Wu *Scribe: Nate Mikosh*

Recap (Dual Regev)

- KeyGen: $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\mathbf{r} \xleftarrow{R} \{0,1\}^m$. $\mathsf{pk} \coloneqq (\mathbf{A}, \mathbf{b} = \mathbf{Ar})$ and $\mathsf{sk} \coloneqq \mathbf{r}$.
- $\mathsf{Encrypt}(pk, \mu)$: $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \xleftarrow{R} \chi^m$ and $e' \xleftarrow{R} \chi$. $\mathsf{ct} \coloneqq (\mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T}, \mathbf{s}^\mathsf{T}\mathbf{b} + e' + \mu\lfloor \frac{q}{2} \rceil)$

Looking at this construction, we can consider the following equation using the first part of the cipher text.

$$\left(s^T A + e^T\right) r = s^T A r + e^T r$$

$e^T r$ is too small to be significant, and $b = Ar$ so we can consider this equation as:

$$\left(s^T A + e^T\right) r = s^T b + noise$$

So effectively, this allows us to take an LWE instance with respect to A and translate it into an instance with respect to b. This is actually a very useful property for encryption, we can use this to switch between keys to create a scheme for ABE. We'll start by looking at a sketch of the construction, then the formal construction, and finally the security proof.

Preliminary Construction Suppose we want to construct ABE for an attribute $\mathbf{x} \in \{0,1\}^\ell$ and a policy $f : \{0,1\}^\ell \to \{0,1\}$ If $f(x) = 0$, we are allowed to decrypt and if $f(x) = 1$, we are not allowed. We will consider key-policy ABE where the secret key is associated with the policy f and the cipher text is associated with the attribute x.

Algebraic Structure In order to create this encryption scheme, we need some way to encode an attribute and then we need to compute on them (homomorphically). Being able to compute on it will allow us to determine if $f(x) = 0$ or 1. Consider the following construction:

pk: $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\mathbf{B} \xleftarrow{R} \mathbb{Z}_q^{n \times m\ell}$

In order to embed the attributes we can use $(\mathbf{B} - \mathbf{x} \otimes \mathbf{G})$ Recall that using the tensor function, this gives us $[\mathbf{B}_1 - x_1\mathbf{G} \mid \cdots \mid \mathbf{B}_\ell - x_\ell\mathbf{G}]$

To enable homomorphic encryption we can use a homomorphic evaluation matrix.

$$(\mathbf{B} - \mathbf{x} \otimes \mathbf{G}) \times \mathbf{H}_{f,\mathbf{x}} = \mathbf{B} \times \mathbf{H}_f - f(x) \times \mathbf{G}$$

When $f(x) = 0$, we are left with $\mathbf{B}_f$ and when $f(x) = 1$ we are left with $\mathbf{B}_f - \mathbf{G}$ and we cannot learn $\mathbf{B}_f$.We can create a ciphertext to be a dual regev encryption with respect to $[\mathbf{A} \mid \mathbf{B}_f]$

Encryption Setup:

$\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$ $\mathbf{B} \xleftarrow{R} \mathbb{Z}_q^{n \times m\ell}$ $u \xleftarrow{R} \mathbb{Z}_q^n$

$[(\mathbf{A}, td\mathbf{A}) \leftarrow trapGen\,(n, q)]$

pk: $(\mathbf{A}, \mathbf{B}, u)$ sk: $td\mathbf{A}$

$Encrypt\,(pk, \mathbf{x}, \mu) : \mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$ $\mathbf{e} \xleftarrow{R} \chi^m$ $e' \xleftarrow{R} \chi$ $\mathbf{R} \xleftarrow{R} \{0,1\}^{m \times m}$

ct: $\left(\mathbf{s}^T\mathbf{A} + e^T, \mathbf{s}^T\,(\mathbf{B} - \mathbf{x} \otimes \mathbf{G}) + e^T\mathbf{R}, \mathbf{s}^T\mathbf{u} + e' + \mu\lfloor\frac{q}{2}\rceil\right)$

KeyGen(sk, f): we only want to recode $\mathbf{u}$ when $f(x) = 0$.

we can try $\mathbf{s^T}\,(\mathbf{B} - \mathbf{x} \otimes \mathbf{G}) + e^T\mathbf{R} - \mathbf{H}_{f,\mathbf{x}} = \mathbf{s}^T\,(\mathbf{B}_f - f(x)\mathbf{G})$

we need to use $[\mathbf{A} \mid \mathbf{B}_f]^{-1}$ for recoding

$z_f = [\mathbf{A} \mid \mathbf{B}_f]^{-1}\,(u)$

There is a new issue now though, that we have a trapdoor for A, but not a trapdoor for this whole matrix. We can change our secret key to instead be a trapdoor $\mathbf{T} \in \mathbb{Z}_q^{m \times m}$ where $\mathbf{AT} = \mathbf{G}$. So now $[\mathbf{A} \mid \mathbf{B}_f] \times \begin{bmatrix}\mathbf{T}\\ 0\end{bmatrix} = \mathbf{G}$

Final Construction Decrypt(sk, ct) where sk = z_f and ct is (c1, c2, c3) : $\lfloor c3 - \left[c1^{\mathbf{T}} \mid c2^{\mathbf{T}}\mathbf{H}_{f,\mathbf{x}}\right] z_f \rceil$

$\left[c1^{\mathbf{T}} \mid c2^{\mathbf{T}}\mathbf{H}_{f,\mathbf{x}}\right] = \mathbf{s^T}\,[\mathbf{A} \mid \mathbf{B}_f - f(x)G] + \left[e^{\mathbf{T}} \mid e^{\mathbf{T}}\mathbf{R}\mathbf{H}_{f,\mathbf{x}}\right]$

$f(x) = 0 : s^{\mathbf{T}}\,[\mathbf{A} \mid \mathbf{B}_f] + \left[e^{\mathbf{T}} \mid e^{\mathbf{T}}\mathbf{R}\mathbf{H}_{f,\mathbf{x}}\right]$

$\times \mathbf{z}_f$

$= \mathbf{s}^{\mathbf{T}}\,[\mathbf{A} \mid \mathbf{B}_f]\,\mathbf{z}_f + \left[e^{\mathbf{T}} \mid e^{\mathbf{T}}\mathbf{R}\mathbf{H}_{f,\mathbf{x}}\right] = \mathbf{s^T}\mathbf{u} + error$

Security Our ciphertext components are similar to LWE, and we can try to prove security by reducing to LWE

In fact, it seems like the only significant difference is that $e^{\mathbf{T}}\mathbf{R}$ in c2 should have a fresh error. Otherwise, this construction would directly follow from LWE. However we cannot actually rely entirely on LWE. ABE also requires there to be key generation and therefore a trapdoor. LWE does not allow for a trapdoor. We need to be able to generate a key for functions that cannot decrypt, and not be able to for those that can. We need possibly exponentially many keys and to not have a trapdoor. We can make this problem simpler by relaxing to selective security, not adaptive security. The adversary will tell us the attribute in advance, but we still need to be able to generate keys for exponentially many functions where $f(x) = 1$. We can consider that we only have to generate a key when $f(x) = 1$ to help us. Suppose we have the following relation:

$\mathbf{ARH}_{f,\mathbf{x}} = \mathbf{B}_f - \mathbf{G}$ $\mathbf{ARH}_{f,\mathbf{x}} - \mathbf{B}_f = -\mathbf{G}\,[\mathbf{A} \mid \mathbf{B}_f] \begin{bmatrix}\mathbf{RH}_{f,\mathbf{x}}\\ -\mathbf{I}\end{bmatrix} = -G$

We have managed to find a trapdoor, not for A, but for $[\mathbf{A} \mid \mathbf{B}_f]$,but only when $f(x) = 1$.

Selective Security Game Adversary starts by committing to an attribute $x^* \in 0, 1^\ell$ and receives in return the public keys, $\mathbf{A}, \mathbf{B}, and\mathbf{u}$. The adversary then sends a function $fwheref(x^*) = 1$ and the challenger will respond with z_f. Next the adversary will send the challenge, a message, μ. The challenger then responds based with either $\left(\mathbf{s}^T\mathbf{A} + e^T, \mathbf{s}^T\,(\mathbf{B} - \mathbf{x} \otimes \mathbf{G}), \mathbf{s}^T\mathbf{u} + e' + \mu\lfloor\frac{q}{2}\rceil\right)$ or simply something uniformly random. The adversary can continuously make key-gen queries after this.

We need to modify this somewhat to arrive at our final proof for security. We can sample $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$ and set $\mathbf{B} = \mathbf{AR} + (x^* \otimes \mathbf{G})$.

In our final game we have an LWE challenger, a reduction, and the adversary. The challenger samples $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^n$ and then either $t_1 = \mathbf{s^T}\mathbf{A} + e^{\mathbf{T}} and t_2 = \mathbf{s^T}\mathbf{u} + e'$ or $t_1 \xleftarrow{R} \mathbb{Z}_q^n and t_2 = \xleftarrow{R} \mathbb{Z}_q^n$. This give us an LWE instance of $(\mathbf{A}, \mathbf{u}, t_1, t_2)$. The adversary again starts by committing to an attribute

$x^* \in 0,1^\ell$ and gets back the public keys,$\mathbf{A}, \mathbf{B} = \mathbf{AR} + (x^* \otimes \mathbf{G}), and \mathbf{u}$. The adversary can then make a keygen query, sending f where $f(x^*) = 1$ and we respond with $z_f = [\mathbf{A} \mid \mathbf{B}_f]^{-1}(u)$. Remember that we can use our trapdoor $\begin{bmatrix} \mathbf{RH}_{f,\mathbf{x}} \\ -\mathbf{I} \end{bmatrix}$ to compute this efficiently. And now for every message μ the adversary sends, we can respond with $t_1^{\mathbf{T}}, t_1^{\mathbf{T}}\mathbf{R}, t_2 + \mu\lfloor\frac{q}{2}\rceil$

Bibliography

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In *PKC*, pages 207–228, 2006.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, pages 213–229, 2001.

[BG04] Daniel R. L. Brown and Robert P. Gallant. The static diffie-hellman problem. *IACR Cryptol. ePrint Arch.*, page 306, 2004.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, pages 416–432, 2003.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, pages 325–341, 2005.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO*, pages 258–275, 2005.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT*, pages 514–532, 2001.

[BLS02] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In *SCN*, pages 257–267, 2002.

[CBBZ22] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. Cryptology ePrint Archive, Paper 2022/1355, 2022. https://eprint.iacr.org/2022/1355.

[Che06] Jung Hee Cheon. Security analysis of the strong diffie-hellman problem. In *EUROCRYPT*, pages 1–11, 2006.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, pages 467–476, 2013.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In *EUROCRYPT*, pages 339–358, 2006.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, pages 89–98, 2006.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, pages 197–206, 2008.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. Cryptology ePrint Archive, Paper 2013/340, 2013. https://eprint.iacr.org/2013/340.

[GWC19] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.

[Jou00] Antoine Joux. A one round protocol for tripartite diffie-hellman. In *ANTS*, pages 385–394, 2000.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In *ASIACRYPT*, pages 407–441, 2023.

[KZG10] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010.

[MVO91] Alfred Menezes, Scott A. Vanstone, and Tatsuaki Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *STOC*, pages 80–89, 1991.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, pages 84–93, 2005.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014.

[WW22] Brent Waters and David J. Wu. Batch arguments for np and more from standard bilinear group assumptions. In *CRYPTO*, pages 433–463, 2022.