This lecture covers:

- Solving low-density subset sum with LLL.
- Coppersmith’s Theorem: finding small roots of polynomials.
- Factoring an RSA modulus knowing a few higher order bits of one of the factors using Coppersmith.

## 1 Solving Low-density Subset Sum

**Definition 1.** Subset sum (SSUM) is the following problem: given $a_1, \ldots, a_n \in [0, X]$ and $s = \sum a_i x_i$ where each $x_i \in \{0, 1\}$, find $\vec{x} = (x_1, \ldots, x_n)$.

**Definition 2.** The density of a subset sum problem is defined as $\frac{n}{\log X}$; the ratio between the number of elements in your sum to the number of bits in the range of $a_i$’s.

Low density means $\frac{n}{\log X}$ is very small, for example $\frac{1}{n^2}$ where $X = 2^{n^2}$.

**Theorem 3** (Frieze). Let $X = 2^{\Omega(n^2)}$. There is an average-case polytime algorithm for SSUM.

**Proof.** We are given $a_1, \ldots, a_n \in [0, X]$, and the sum $s = \sum_{i=1}^{n} a_i x_i$, where each $x_i \in \{0, 1\}$. First, we are going to phrase this as an SVP in a lattice. We define a lattice

$$
\mathcal{L}_{a_1, \ldots, a_n, s} = \begin{bmatrix}
I & 0 \\
\vdots & \vdots \\
a_1 & a_2 & \ldots & a_n & s
\end{bmatrix}
$$

in $n + 1$ dimensions. Notice that if we make a column vector of the $x_i$, we get

$$
\begin{bmatrix}
1 & 0 & \ldots & 0 \\
0 & 1 & \ldots & 0 \\
\vdots & \ddots & \ddots & \vdots \\
a_1 & a_2 & \ldots & a_n & s
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n
\end{bmatrix} = \begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n
\end{bmatrix},
$$

and only a solution to the subset sum problem will have this property. So, a SSUM solution is a lattice vector of length $\sqrt{n}$ such that

$$
\mathcal{L} \cdot \begin{bmatrix}
x \\
-1
\end{bmatrix} = \begin{bmatrix}
x \\
0
\end{bmatrix}.
$$

We want to guarantee that the only small solutions are of the form $\alpha x$ – it is easy to find $\alpha$ if we know $x$, so we will scale each $a_i$ and the sum $s$ in the basis by some large $\beta = 2^{\Omega(n)}$. The problem then becomes finding
a vector $z$ of dimension $n + 1$ such that

$$
\begin{bmatrix}
I \\
\vdots \\
0
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
\vdots \\
an \\
0 \\
s
\end{bmatrix}
\begin{bmatrix}
z_1 \\
vdots \\
z_n \\
z_{n+1}
\end{bmatrix}
=
\begin{bmatrix}
z_1 \\
vdots \\
z_n \\
0
\end{bmatrix}.
$$

Claim 4. With high probability, the only small solutions are $\alpha \cdot \begin{bmatrix} x \\ -1 \end{bmatrix}$.

Proof. We start with $\sum_{i=1}^{n} \beta a_i z_i + \beta z_{n+1} s = 0$, and we can divide out by $\beta$ to get $\sum_{i=1}^{n} a_i z_i + z_{n+1} s = 0$. We also have that $\sum_{i=1}^{n} a_i x_i - s = 0$ from the original solution. For $i = 1, \cdots, n$, let $y_i = x_i - z_i$ and $y_{n+1} = z_{n+1} - 1$. Subtracting one from the other, we have

$$\sum_{i=1}^{n} a_i (x_i - z_i) - (z_{n+1} - 1) s = \sum_{i=1}^{n} a_i y_i - y_{n+1} s = 0.$$

Now, notice two things

1. First, fix the $y_i$, and we have $\Pr(a_i | \sum a_i y_i - y_{n+1} s = 0) = \frac{1}{X}$.

2. Now, we note that the number of possible $y_i$’s is small, $2^{O(n^2)}$, based on the approximation LLL outputs.

So,

$$\Pr(\sum a_i y_i - y_{n+1} s = 0 \text{ for some } y \neq 0) = \frac{1}{X} \cdot 2^{O(n^2)}.$$

Since $X = 2^{O(n^2)}$, $\frac{1}{X} \cdot 2^{O(n^2)}$. We can run the LLL algorithm for approximating the shortest vector. The output vector, $z$, is guaranteed to be a $2^{O(n)}$-approximate shortest vector. From the claim, we know that $z$, with high probability, is of the form $\alpha \begin{bmatrix} x \\ -1 \end{bmatrix}$. Finding $x$ from the product is easy; since each $x_i \in \{0, 1\}$, we know the value of $\alpha$. \qed

2 Coppersmith and Applications

Theorem 5 (Coppersmith). There is a poly(log $N, d$)-time algorithm that given $f(x) \in \mathbb{Z}[x]$, a degree $d$ monic polynomial, outputs all $x_0$ such that

- $f(x_0) = 0 \mod N$
- $|x_0| < N^{1/d}$.

Note: this implies that there are polynomially many small roots mod $N$!

Example 1. Consider the polynomial $x^3 - a = 0 \mod N$. We want to find all roots $|x_0| < N^{1/3}$. We notice that $|x_0| < N^{1/3}$ implies $x_0^3 < N$. This implies $x_0^3 = a$ over $\mathbb{Z}$. We have reduced the problem to finding cube roots over $\mathbb{Z}$!

Proof. So, let $f$ be any monic polynomial over $\mathbb{Z}$, degree $d$, and $B = N^{1/d}$. We can represent $f(x) = \sum_{i=0}^{d} f_i x^i$ (note that $f_d = 1$). From $f$, we will define $h(x) = \sum h_i x^i$ so that
• All roots $x_0$ of $f(x) \mod N$ are also roots of $h(x)$.

• $|h_i B^i| < \frac{N}{d+1}$.

This implies that for every root $x_0$, $|h(x_0)| \leq |h(B)| \leq \sum h_i B^i < N$. So, we will have reduced the problem to finding roots of $h$ over $\mathbb{Z}$.

To find $h$, we start with a basis set of size $d + 1$: $\{N, N x, \ldots, N x^d\}$. We will let our basis

$$B = \begin{bmatrix}
N & 0 & 0 & \ldots & 0 & f_0 \\
0 & BN & 0 & \ldots & 0 & f_1 B \\
0 & 0 & B^2 N & \ldots & 0 & f_2 B^2 \\
0 & 0 & 0 & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & B^{d-1} N & f_{d-1} B^{d-1} \\
0 & 0 & 0 & \ldots & 0 & B^d \\
\end{bmatrix},$$

where the rightmost column of $B$ are the coefficients of $f(B x)$, and the diagonal is $B N$.

If we run LLL on $L(\mathbb{B})$, then we get an approximate small vector $(v_0, v_1, \ldots, v_d)$. We notice that each coordinate $v_i$ of $v$ is divisible by $B^i$, from our basis. Thus, we can define the integer coefficients of $h$ as $h_i = v_i / B^i$. Now, by construction, for every $x_0$ such that $f(x_0) = 0 \mod N$, $h(x_0) = 0$. Finally, we need to show that $|h_i B^i| < \frac{N}{d+1}$.

Recall that LLL is a $2^{d+1}$ approximation, and that Minkowski’s bound tells us that $\lambda_1 \leq \sqrt{d + 1} \det(\mathbb{B})^{1/(d+1)}$. The magnitude of $v$ is

$$||v|| \leq 2^{d+1} \sqrt{d + 1} \det(\mathbb{B}) = 2^{d+1} \sqrt{d + 1} (N^d \cdot B^{d(d+1)/2})^{1/d} = 2^{d+1} \sqrt{d + 1} N^{d/(d+1)} B^{d/2} = c_d N^{d/(d+1)} B^{d/2},$$

where $c_d = 2^{d+1} \sqrt{d + 1}$ is a constant only dependent on $d$. Also, $\frac{d}{d+1} = 1 - \frac{1}{d+1}$, so if we take $B$ small enough,

$$h_i B^i = |v| \leq ||v|| \leq c_d B^{d/2} N^{1-1/(d+1)} < \frac{N}{d+1}.$$

We can then factor $h$ over $\mathbb{Z}$ to get the roots of $f$ over $\mathbb{Z}_N$.

### 2.1 Factoring with a few known bits

The goal will be to break RSA in a modulus $N = pq$ when we are given half of the bits of $p$, $1/2 \log p$ bits, in $\text{poly}(\log N)$ time. Before Coppersmith’s algorithm, Rivest and Shamir were able to find $p$ with $2/3 \log p$ bits.

**Theorem 6.** Given $N = pq$, $p \approx N^{\gamma}$ where $\gamma \geq 2/3$, and $\tilde{p}$ is half of the bits of $p$, we can find all of $p$ in $\text{poly}(\log N)$ time.

**Proof.** Given $\tilde{p}$, we let $f(x) = x + \tilde{p}$. Our goal will be to find a root of $f(x) \mod p$ without prior knowledge of $p$. We will define a bound $B < N^{1/3}$ to use in Coppersmith’s algorithm. We get the following 2-dimensional basis:

$$\mathbb{B} = \begin{bmatrix} N & \tilde{p} \\
0 & B \end{bmatrix}.$$

In this lattice, Minkowski’s bound tells us that $\lambda_1 \leq \det(\mathbb{B})^{1/2} = \sqrt{NB}$. Running LLL on $\mathbb{B}$ gives us a small vector $v = \begin{bmatrix} h_0 \\
B h_1 \end{bmatrix}$. Since LLL finds a $2^d$-approximate small vector (and $d = 2$), $||v|| \leq 2\sqrt{NB}$. We wanted to define $B$ so that the LLL approximation gives us a small enough vector. So, we need $||v|| < p \approx N^{2/3}$, meaning $2\sqrt{NB} < N^{2/3}$. If we let $B < N^{1/3}$, this inequality holds.

So, for any $x_0 < B$ in $\mathbb{Z}$, $h(x_0) = h_0 + h_1 x_0 \leq ||v|| \leq 2\sqrt{NB}$. Now, consider $x_0 < B$ an integral root of $h$. Since $B < p$, $x_0$ is a root of $h \mod p$. $|x_0| < \tilde{p}$, so $f(x_0) = x_0 + \tilde{p} \equiv 0 \mod p$, meaning $\gcd(f(x_0), N) = p$. We have found the rest of the bits of $p$! \[\square\]
2.2 Attacks on padding in low exponent RSA

Recall how RSA works. A modulus $N = pq$ (usually on the order of 2000 bits) and a public key $e$ are public. The decryption key, $d = e^{-1} \mod \phi(N)$, is private. For Alice to send a message $M$ to Bob, she computes $C = f(M) = M^e \mod N$. Bob, with his private key, can decrypt $C$: $C^d = M^{ed} \mod N = M \mod N$.

Notice this is a deterministic scheme, so an attacker can guess at what message is being sent and check by encrypting his guess against the original message.

A common defense against this kind of attack is to pad the message with random bits. So, for a message $M \in \{0, 1\}^n$, we encrypt by finding a random $r \in \{0, 1\}^m$ and letting our ciphertext $C = f(M \| r)$. Mathematically, we are taking $M, r \in \mathbb{Z}_N$, and letting $M' = 2^m M + r$. We will soon show how this kind of padding offers no security.

**Lemma 7.** Let $e = 3$ and $\ell(x) = ax + b$ for $a, b \neq 0$. Given the RSA public parameters $e, N$ and two ciphertexts $C_1, C_2 \in \mathbb{Z}_N^*$, where $C_1 = f(M_1)$ and $C_2 = f(M_2)$ for messages $M_1, M_2$ so that $M_1 = \ell(M_2)$, we can find both $M_1$ and $M_2$ efficiently.

**Proof.** Let $g_1(x) = \ell(x)^e - C_1$ and $g_2(x) = x^e - C_2$. Notice that $M_2$ is a root of both $g_1$ and $g_2$. If we can prove that $(x - M_2)$ is the gcd of $g_1$ and $g_2$, then we can easily compute $(x - M_2)$ using the Euclidean algorithm on $g_1$ and $g_2$.

Recall that RSA is a bijection, so there is only one root in $\mathbb{Z}_N$ of $g_2$, and that root is $M_2$. So, $g_2(x) = (x - M_2)g'(x)$ where $g'$ is a quadratic irreducible in $\mathbb{Z}_N$. So, $\gcd(g_1, g_2) = (x - M_2)$ or $g_2$. However, since $b \neq 0$, $M_1 \neq M_2$, so $g_2 \nmid g_1$. Therefore $\gcd(g_1, g_2) = (x - M_2)$. \hfill $\square$

**Theorem 8.** Let $N \approx 2^n$ be an RSA modulus, $e = 3$, and the padding length $m \leq \lfloor n/e^2 \rfloor$. Given $C_1 = f(M \| r_1)$ and $C_2 = f(M \| r_2)$, where $r_1 \neq r_2$, we can recover $M$ efficiently.

**Proof.** Let’s define $M_1 = 2^m M + r_1$ and $M_2 = 2^m M + r_2$. Our goal will be to determine $M$ and $r_1$ and $r_2$. So, let’s let $x$ be our unknown message and $y$ be our unknown padding. Based on these variables, we define $g_1(x, y) = x^e - C_1$ and $g_2(x, y) = (x + y)^e - C_2 = (x + y)^e - M_2^e$.

Since RSA is a bijection, $g_1$ implies that $x = M_2$. Given that $x = M$, $g_2$ implies that $y = r_2 - r_1$.

Next, we want to consider the **resultant** of $g_1$ and $g_2$. The resultant on two polynomials $p(x)$ and $q(x)$ is defined as $\text{res}_x(p(x), q(x)) = \prod_{p(x_1) = q(x_2) = 0} (x_1 - x_2)$.

There are a couple of things we can note about the resultant:

- If $p$ and $q$ share a root, then $\text{res}_x(p(x), q(x)) = 0$.
- $\text{res}_x(p, q)$ is also the determinant of the Sylvester matrix of $p$ and $q$, $S_{p, q}$. Therefore, it can be computed efficiently.

We will want to solve for $y$ first, so we compute the resultant of $g_1$ and $g_2$ based on the $x$-coefficients of $y$. Notice that $g_1(x, y)$ is degree 0 with respect to $y$ and that $g_2(x, y)$ is degree $e = 3$, so $\text{res}_x(g_1, g_2)$ has degree at most $e^2$ in $y$.

Let $h(y) = \text{res}_x(g_1, g_2)$. Notice that $\Delta = r_2 - r_1$ is a root of $h$, since setting $y$ to $\Delta$ makes $M_1$ a root of both $g_1$ and $g_2$. We also have that $\Delta$ is small; $|\Delta| < 2^m < N^{1/e^2}$. So, we can run Coppersmith’s root-finding algorithm to get a polynomial list of candidate $\Delta$s.

For each candidate $\Delta$, we let $\ell = x - \Delta$ and use the algorithm in lemma 7, revealing candidates $M_1$ and $M_2$. We check if we are successful by re-encrypting them to see if they are equal to $C_1$ and $C_2$. \hfill $\square$
References
