9 Knapsack Cryptography

In the past four weeks, we've discussed public-key encryption systems that depend on various problems that we believe to be hard: prime factorization, the discrete logarithm, and the elliptic curve discrete logarithm. These algorithms currently work fine in practice, but they have three potential security problems:

1. We don't know for sure that it's possible to have a secure public-key type algorithm; anything that can efficiently solve NP-complete problems can efficiently break public-key cryptography.

2. We don't actually know that the discrete logarithm class of problems is NP-complete; it is suspected that it is not. Thus it's possible that these problems can be efficiently solved even if $P \neq NP$.

3. There is a known *quantum* algorithm that can efficiently break all of these cryptosystems. Because building a functioning quantum computer is a challenging engineering task, the largest number that has been factored with Shor's algorithm so far is 21. Thus quantum computers are not currently a threat.

   But there is a substantial risk of them becoming a threat in the medium future, so many security-concerned groups have started planning for "post-quantum" cryptography that can stand up to quantum computer based attacks.

We can't really do anything about the first problem; if $P = NP$ then public-key cryptography is impossible in principle. But the other two problems are potentially solvable, by basing our cryptographic algorithm on a problem that is "known" to be hard—a problem that is NP-complete.

9.1 The Subset-Sum Problem

A classic NP-complete problem is the *knapsack problem*, which asks you to maximize the value of items you can fit into a knapsack subject to some weight restrictions.

Definition 9.1 (Knapsack problem)**.** Given a (finite) set of items x_i, each with weight w_i and value v_i, maximize $\sum_{i=1}^{n} x_i v_i$ subject to $\sum_{i=1}^{n} x_i w_i \leq W, x_i \in \{0, 1\}$.

A special case of this is the subset sum problem

Definition 9.2 (Subset Sum problem)**.** Given a list of positive integers $\mathbf{M} = (M_1, \dots, M_n)$, and another integer S, find a subset of the integers in the list whose sum is S.

(This corresponds to the knapsack problem with each $v_i = w_i = M_i$).

This problem is known to be NP-complete in general: it is easy to check that a possible solution is an actual solution (by adding the numbers and checking that they sum to S), but difficult to find such a subset if you don't already know one.

Example 9.3. Solve the subset-sum problem for the list $(1, 3, 5, 6, 8, 10, 11)$ and the sum $S = 24$.

A little trial and error gives that we have $S = 3 + 10 + 11$. Alternatively $S = 6 + 8 + 10$. There's no guarantee that any given solution is unique.

How *do* we solve the subset-sum problem? We can obviously brute-force the solution by trying all 2^n possible combinations. A collision algorithm like the Giant Step/Baby Step algorithm can reduce the number of calculations necessary by a square root, so we can solve the problem in $\mathcal{O}(2^{n/2+\epsilon})$ steps. This is still quite slow.

We'd like to use the one-way difficulty of solving this problem to power a cryptosystem. The basic idea is that Alice starts with a list of positive integers $\mathbf{M} = (M_1, \dots, M_n)$. She writes her message as an n-bit binary number with digits $x_1 x_2 \dots x_n$, and then sends Bob the number $C = \sum_{i=1}^{n} x_i M_i$. If Bob can solve the subset-sum problem, he can recover the list of integers M_i and thus the original message $M = x_1 x_2 \dots x_n$.

If Eve intercepts this message, she will have a difficult time decrypting it, because solving the subset sum problem is difficult. But without any extra tools, Alice can't decrypt it any more easily than Eve can! So we need some sort of "trap door" that allows Alice to solve the subset-sum problem easily. So our goal is to set up a specific subset-sum problem that is easy to solve, and then somehow disguise it so no one else can solve it easily.

Definition 9.4. A list of positive integers $\mathbf{r} = (r_1, \dots, r_n)$ is a *superincreasing sequence* if $r_{i+1} \geq 2r_i$ for all i.

Lemma 9.5. *If $\boldsymbol{r} = (r_1, \dots, r_n)$ is a superincreasing sequence, then $r_k > r_{k-1} + \dots + r_2 + r_1$ for all $2 \leq k \leq n$.*

Proof. We prove this with a straightforward induction. For $k = 2$ we have $r_2 \geq 2r_1 > r_1$. Suppose that $r_k > r_{k-1} + \dots + r_1$. Then we have

$$r_{k+1} \geq 2r_k = r_k + r_k > r_k + (r_{k-1} + \dots + r_1)$$

as desired.

□

The subset-sum problem is very easy to solve for a superincreasing sequence.

Proposition 9.6. *Let $(\boldsymbol{M}, S)$ give a subset-sum problem, where the integers in $\boldsymbol{M} = (M_1, \dots, M_n)$ form a superincreasing sequence. If a solution $\boldsymbol{x}$ exists, we may find it with the following algorithm:*

1. *Start with the largest element M_n.*
2. *If $S \geq M_i$, set $x_i = 1$ and subtract M_i from S. Otherwise, set $x_i = 0$.*
3. *Proceed to the next smallest number.*

At the end of this process, $\boldsymbol{x} = x_1 \dots x_n$ is a solution to the subset sum problem.

Proof. Suppose we have a solution $\mathbf{y}$, such that $\mathbf{y} \cdot \mathbf{M} = S$. We want to show that the number $\mathbf{x}$ produced by the algorithm is equal to $\mathbf{y}$. We prove this by downward induction.

Suppose that $x_i = y_i$ for all $k < i \leq n$; we wish to prove that $x_k = y_k$. We have

$$S_k = S - \sum_{i=k+1}^{n} x_i M_i = \sum_{i=1}^{n} y_i M_i - \sum_{i=k+1}^{n} x_i M_i = \sum_{i=1}^{n} y_i M_i - \sum_{i=k+1}^{n} y_i M_i = \sum_{i=1}^{k} k y_i M_i$$

When we execute the loop for $i = k$, there are two possibilities.

1. If $y_k = 1$, then $S_k \geq M_k$, so we set $x_k = 1$. So $x_k = y_k$.
2. If $y_k = 0$, then $S_k \leq M_{k-1} + \dots + M_1 < M_k$ by lemma 9.5, and thus we set $x_k = 0$ in our algorithm, and so $y_k = x_k$.

Thus in either case, $x_k = y_k$, and thus by induction $x_i = y_i$ for all i.

Thus if a solution exists, our algorithm finds it. This proves that our algorithm works, and also that the solution is unique.

□

Example 9.7. The set $\mathbf{M} = (3, 11, 24, 50, 115)$ is superincreasing. Solve the subset-sum problem for $(\mathbf{M}, 142)$.

We see that $S \geq 115$, so we have $x_5 = 1$ and $S_5 = 142 - 115 = 27$. Then $S_5 < 50$ so $x_4 = 0$ and $S_4 = 27$. $S_2 > 24$ so $x_3 = 1$ and $S_3 = 27 - 24 = 3$. Then $S_3 < 11$ so we have $x_2 = 0$ and $S_2 = 3$. Finally we have $S_2 \geq 3$ so $x_1 = 1$ and $S_1 = 0$, which it necessarily will.

Thus our algorithm claims that

$$142 = 1 \cdot 3 + 0 \cdot 1 + 1 \cdot 24 + 0 \cdot 50 + 1 \cdot 115$$

which is in fact true.

9.2 Knapsack Cryptography

Merkle and Hellman proposed a public key system based on a super-increasing subset-sum problem. In order to make it difficult for Eve to decipher, the superincreasingness is disguised using congruences. Since Alice knows how to use the superincreasingness, she can solve the problem easily; since Eve does not, she cannot.

Algorithm 9.1 (Merkle-Hellman Subset-Sum Cryptography)**.** To create the public-private keypair:

1. Alice chooses a superincreasing sequence of positive integers $\mathbf{r} = (r_1, \dots, r_n)$.
2. Alice chooses two large integers A, B with $B > 2r_n$ and $\gcd(A, B) = 1$. Alice also computes the inverse of A modulo B.
3. For each i, Alice sets $M_i \equiv Ar_i \mod B$ with $0 \leq M_i < B$. The sequence $\mathbf{M} = (M_1, \dots, M_n)$ is Alice's public key, which she publishes.

When Bob wants to encrypt a message:

1. Bob writes his plaintext $\mathbf{x}$ as a binary vector.
2. Bob computes $S = \mathbf{x} \cdot \mathbf{M} = \sum_{i=1}^{n} x_i M_i$ and sends this to Alice.

To decrypt:

1. Alice computes $S' \equiv A^{-1}S \mod B$ for $0 \leq S' < B$.
2. Alice solves the subset problem for $(\mathbf{r}, S')$. Since $\mathbf{r}$ is superincreasing, this is easy.

The decryption works becasue we have

$$\begin{aligned} S' &\equiv A^{-1}S \equiv A^{-1} \sum_{i=1}^{n} x_i M_i \\ &\equiv A^{-1} \sum_{i=1}^{n} x_i A r_i \equiv \sum_{i=1}^{n} x_i r_i \mod B. \end{aligned}$$

Since $B > 2r_n$ we know that $B > \sum_{i=1}^{n} x_i r_i$, so we know that S' is in fact an exact sum of the r_i and Alice can solve her problem easily.

Example 9.8. Suppose Alice chooses $\mathbf{r} = (3, 11, 24, 50, 115)$ and she chooses $A = 113, B = 250$. Then she computes

$$\begin{aligned}\mathbf{M} &= (113 \cdot 3, 113 \cdot 11, 113 \cdot 24, 113 \cdot 50, 113 \cdot 115) \mod 250 \\ &= (89, 243, 212, 150, 245).\end{aligned}$$

She also computes that $A^{-1} \equiv 177 \mod 250$.

Bob wants to send Alice the message $\mathbf{x} = (1, 0, 1, 0, 1)$ (which corresponds to the number 21 in binary). He encrypts $\mathbf{x}$ by computing

$$S = \mathbf{x} \cdot \mathbf{M} = 1 \cdot 89 + 0 \cdot 243 + 1 \cdot 212 + 0 \cdot 150 + 1 \cdot 245 = 546$$

and sending this to Alice.

When Alice receives S, she computes $177 \cdot 546 \equiv 142 \mod 250$. She can then solve the subset-sum problem (as we did in example 9.7) to recover Bob's message.

9.2.1 Parameter sizes and security

If our subset has n elements, then there are 2^n possible binary vectors, so a brute force attempt would invole 2^n calculations. Because there is a collision algorithm, we can do things in $2^{n/2}$ calcuations, so achieving 80 bits of security reuqires taking $n > 160$.

It turns out we also need to choose $r_1 > 2^n$, which implies that $r_n > 2^{2n}$ and thus $B > 2^{2n}$ Thus the public key is a list of n integers, each of which is approximately 2^n bits long; the plaintext contains n bits of information, and the ciphertext is a number of approximately $2n$ bits. This gives us again a 2-1 message expansion factor.

Notice that the public keys are really long! To get an effective 80 bits of security, elliptic curve cryptography required a 160-bit key; RSA required approximately a 1024-bit key. Our knapsack cryptosystem here requires a public key of size $2n^2 = 51200$ bits.

This seems like a problem, but it's compensated for by the fact that knapsack cryptosystems are extremely fast. Notice that our cryptosystem involved no multiplication to encrypt, and only one multiplication to decrypt; this is far more efficient than the repeated exponentiations involved in RSA, Diffie-Hellman, and ECC.

So why aren't knapsack cryptosystems the new standard? We observed that the subset-sum problem is NP-complete, and in fact the best known algorithms to solve an *arbitrary* subset-sum problem are versions of the collision algorithm, which run in $\mathcal{O}(2^{n/2})$.

But just as using a superincreasing set makes it easy for Alice to decrypt this message, it also adds structure to the message that allows Eve to decrypt it—even without knowing

the original sequence. To do this, Eve reformulates the subset-sum problem as a question about *lattices*.

9.3 Lattices

Definition 9.9. Let $\mathbf{v}_1, \dots, \mathbf{v}_n \in \mathbb{R}^m$ be a set of linearly independent vectors. Then the *lattice L generated by* $\{\mathbf{v}_1, \dots, \mathbf{v}_n\}$ is the set of integeger linear combinations of the $\mathbf{v}_i$ with coefficients in $\mathbb{Z}$:

$$L = \{a_1\mathbf{v}_1 + \cdots + a_n\mathbf{v}_n : a_i \in \mathbb{Z}\}.$$

A lattice is an attempt to take linear algebra, but use the integers as scalars. This changes a number of things because we can't divide by a scalar. There's a lot of theory of lattices we could cover; there's no way to fit it all into one lecture. We'll just be going over some highlights here.

Definition 9.10. An *integral lattice* is a lattice all of whose vectors have integral coordinates. We can view such a lattice as a *subgroup* of $\mathbb{Z}^m$.

Example 9.11. Consider $L \subset \mathbb{R}^3$ generated by $\mathbf{v}_1 = (2, 1, 3), \mathbf{v}_2 = (1, 2, 0), \mathbf{v}_3 = (2, -3, -5)$.

We can define new vectors $\mathbf{w}_1 = \mathbf{v}_1 + \mathbf{v}_3, \mathbf{w}_2 = \mathbf{v}_1 - \mathbf{v}_2 + 2\mathbf{v}_3, \mathbf{w}_3 = \mathbf{v}_1 + 2\mathbf{v}_2$. This gives the change of basis matrix

$$U = \begin{bmatrix} 1 & 0 & 1 \\ 1 & -1 & 2 \\ 1 & 2 & 0 \end{bmatrix}.$$

We can compute that this matrix has determinant -1, so the vectors $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$ are also a basis for L. The inverse of U is

$$U^{-1} = \begin{bmatrix} 4 & -2 & -1 \\ -2 & 1 & 1 \\ -3 & 2 & 1 \end{bmatrix}$$

which tells us we have $\mathbf{v}_1 = 4\mathbf{w}_1 - 2\mathbf{w}_2 - \mathbf{w}_3, \mathbf{v}_2 = -2\mathbf{w}_1 + \mathbf{w}_2 + \mathbf{w}_3, \mathbf{v}_3 = -3\mathbf{w}_1 + 2\mathbf{w}_2 + \mathbf{w}_3$.

Finally, we can ask how to represent the $\mathbf{w}_i$ in "true" coordinates. We can write down the matrix A whose rows are the $\mathbf{v}_i$:

$$\begin{bmatrix} 2 & 1 & 3 \\ 1 & 2 & 0 \\ 2 & -3 & -5 \end{bmatrix}.$$

Then the matrix

$$B = UA = \begin{bmatrix} 4 & -2 & -2 \\ 5 & -7 & -7 \\ 4 & 5 & 3 \end{bmatrix}$$

is the matrix formed by the $\mathbf{w}_i$, so we have $\mathbf{w}_1 = (4, -2, 2), \mathbf{w}_2 = (5, -7, -7), \mathbf{w}_3 = (4, 5, 3)$.

Fact 9.12. *Any two bases for a lattice L are related by a matrix having integer coefficients whose inverse also has integer coefficients. Thus the matrix must have determinant ± 1.*

We say such a matrix is an element of the general linear group over the integers $GL_n(\mathbb{Z})$.

Definition 9.13. Let L be a lattice of dimension n with basis $B = \{\mathbf{v}_1, \dots, \mathbf{v}_n\}$. Then the *fundamental dommain* for L corresponding to B is the set

$$\mathcal{F}(B) = \{t_1\mathbf{v}_1 + t_2\mathbf{v}_2 + \cdots + t_n\mathbf{v}_n : 0 \leq t_i < 1\}.$$

Fact 9.14. *Let $L \subset \mathbb{R}^n$ with basis B and let $\mathcal{F}$ be the fundamental domain for L corresponding to B. Then every vector $\mathbf{w} \in \mathbb{R}^n$ can be written uniquely as $\mathbf{w} = \mathbf{t} + \mathbf{v}$ for some $\mathbf{t} \in \mathcal{F}$ and $\mathbf{v} \in L$.*

Definition 9.15. Let L be a lattice of dimension n and let $\mathcal{F}$ be a fundamental domain for L. Then the n-dimensional volume of $\mathcal{F}$ is called the *determinant of L* or $\det(L)$.

Fact 9.16. *Let L be a lattice, $B = \{\mathbf{v}_1, \dots, \mathbf{v}_n\}$ a basis, and $\mathcal{F}$ a fundamental domain. Then*

$$\det L \leq \|\mathbf{v}_1\| \cdot \cdots \cdot \|\mathbf{v}_n\|.$$

Further if we let A be the matrix whose rows are given by the $\mathbf{v}_i$, then $\det L = |\det A|$.

9.4 The Shortest Vector Problem

A standard question is to find the shortest vector contained in a lattice. In a vector space this question makes no sense, since we can always divide a vector by some scalar and get a shorter vector; in a lattice this is not possible.

Definition 9.17 (Shortest-Vector Problem)**.** Given a lattice L with a basis $\{\mathbf{v}_1, \dots, \mathbf{v}_n\}$, find a shortest non-zero vector $\mathbf{v} \in L$.

Many lattices have more than one shortest vector (e.g. the lattice $\mathbb{Z}^2$ contains $(\pm 1, 0)$ and $(0, \pm 1)$ which are all equal lengths; thus each is "a" shortest vector).

The shortest-vector problem is a classic NP-complete problem, so there is no efficient algorithm to solve it in general. (In fact even the best known algorithms are quite bad; there are exponential-time algorithms, but these also require exponential amounts of memory).

We can also pose the related closest-vector problem:

Definition 9.18 (Closest-Vector Problem)**.** Given a lattice L and a vector $\mathbf{w} \in \mathbb{R}^m$ that is not in L, find a vector $\mathbf{v} \in L$ that is closest to $\mathbf{w}$. That is, find a vector $\mathbf{v} \in L$ that minimizes $\|\mathbf{w} - \mathbf{v}\|$.

These problems are both quite difficult to solve exactly; they are somewhat easier to solve approximately, but only somewhat. We can, however, lay out some guidelines for what a solution will look like.

Fact 9.19 (Hermite's Theorem)**.** *Every lattice L of dimension n contains a nonzero vector $\mathbf{v} \in L$ satisfying $\|\mathbf{v}\| \leq \sqrt{n}\det(L)^{1/n}$.*

Fact 9.20 (Gaussian Heuristic)**.** *Let L be a lattice of dimension n. The* Gaussian expected shortest length *is*

$$\sigma(L) = \sqrt{\frac{n}{2\pi e}}(\det L)^{1/n}.$$

The Gaussian heuristic *says that a shortest nonzero vector in a "random" lattice will satisfy $\|\mathbf{v}\| \approx \sigma(L)$.*

This heuristic is just an average. The shorter the actual shortest vector is compared to the expected shortest vector, the easier it is to solve the shortest-vector problem.

9.5 Cracking the knapsack with lattices

We want to reformulate Eve's problem in cracking the knapsack cryptosystem 9.1 as a question about lattices. Suppose she wants to write S as a subset-sum from the set $\mathbf{M} = (m_1, \ldots, m_n)$. She can write a matrix

$$\begin{bmatrix} 2 & 0 & 0 & \ldots & 0 & m_1 \\ 0 & 2 & 0 & \ldots & 0 & m_2 \\ 0 & 0 & 2 & \ldots & 0 & m_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 2 & m_n \\ 1 & 1 & 1 & \ldots & 1 & S \end{bmatrix}$$

We think of each row of this matrix as forming the vector $\mathbf{v}_i$, so that we have

$$\begin{aligned}
\mathbf{v}_1 &= (2, 0, 0, \ldots, 0, m_1) \\
\mathbf{v}_2 &= (0, 2, 0, \ldots, 0, m_2) \\
\vdots &\quad \vdots \\
\mathbf{v}_n &= (0, 0, 0, \ldots, 2, m_n) \\
\mathbf{v}_{n+1} &= (1, 1, 1, \ldots, 1, S).
\end{aligned}$$

Eve will consider the lattice L generated by the $\mathbf{v}_i$:

$$L = \{a_1\mathbf{v}_1 + \cdots + a_n\mathbf{v}_n + a_{n+1}\mathbf{v}_{n+1} : a_i \in \mathbb{Z}\}.$$

Suppose $\mathbf{x} = (x_1, \ldots, x_n)$ is a solution to the subset problem. Then L contains the vector

$$\mathbf{t} = \sum_{i=1}^{n} x_i\mathbf{v}_i - \mathbf{v}_{n+1} = (2x_1 - 1, 2x_2 - 1, \ldots, 2x_n - 1, 0)$$

where the last coordinate is 0 since $S = x_1m_1 + \cdots + x_nm_n$.

Now since $x_i \in \{0, 1\}$, we know that $2x_i - 1 = \pm 1$, and thus $\|\mathbf{t}\| = \sqrt{n}$. But since $m_i = \mathcal{O}(2^{2n})$ and $S = \mathcal{O}(2^{2n})$, we see that $\|\mathbf{v}_i\| = \mathcal{O}(2^{2n}) \gg \sqrt{n}$.

Thus we see that $\mathbf{t}$ is an atypically short vector in L, so solving the subset-sum problem is reducible to the problem of finding a short vector in this lattice.

In fact, we can compute the Gaussian heuristic. We know that $S = \mathcal{O}(2^{2n})$, so $S^{1/n+1} \approx 4$. Thus we see that

$$\begin{aligned}
\sigma(L_{\mathbf{M},S}) &= \sqrt{\frac{n+1}{2\pi e}} \left(\det L_{\mathbf{M},S}\right)^{1/(n+1)} \\
&= \sqrt{\frac{n+1}{2\pi e}} \left(2^n S\right)^{1/(n+1)} \\
&\approx \sqrt{\frac{n+1}{2\pi e}} 8 \approx 1.936\sqrt{n}
\end{aligned}$$

so $\mathbf{t}$ is about half the length of the "expected" shortest vector. (We can improve this even further by multiplying S and the M_i by some large constant C; this will inflate the expected shortest vector by a factor of $\sqrt[n+1]{C}$ but will leave $\mathbf{t}$ unchanged).

Now, we're looking for the shortest vector in L, which we know is difficult—so we've replaced one hard problem with another. However, clever algorithms can find an *approximate* solution in polynomial time. The Lenstra-Lenstra-Lovász (LLL) algorithm can find a vector

with length at most $2^{(n-1)/2}$ times the length of the shortest vector. Since we expect $\mathbf{t}$ to be much shorter than all the other vectors in the lattice, solving this approximate problem is good enough, and thus we can break the cryptosystem.