CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Lecture 24
December 4, 2017

Locked Boxes

Locked boxes

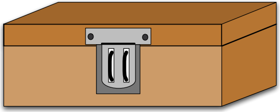
Protocols for coin flipping and for dealing a poker hand from a deck of cards can be based on the intuitive notion of locked boxes.

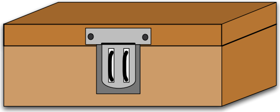
This idea in turn can be implemented using commutative-key cryptosystems.

We first present a coin-flipping protocol using locked boxes.

Preparing the boxes

Imagine two sturdy boxes with hinged lids that can be locked with a padlock.





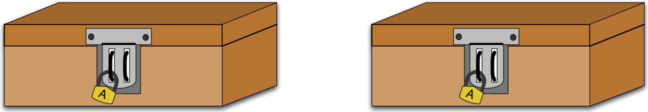
Alice writes “heads” on a slip of paper and “tails” on another.

“heads”, signed Alice	“tails”, signed Alice

She places one of these slips in each box.

Alice locks the boxes

Alice puts a padlock on each box for which she holds the only key.

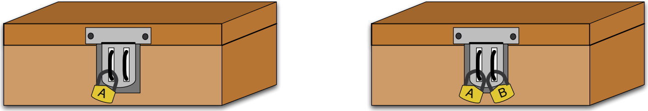


She then gives both locked boxes to Bob, in some random order.

Bob adds his lock

Bob cannot open the boxes and does not know which box contains "heads" and which contains "tails".

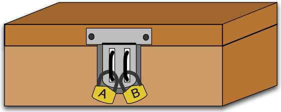
He chooses one of the boxes and locks it with his own padlock, for which he has the only key.



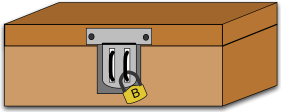
He gives the doubly-locked box back to Alice.

Alice removes her lock

Alice gets



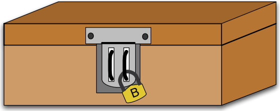
She removes her lock.



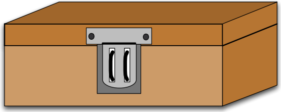
and returns the box to Bob.

Bob opens the box

Bob gets



He removes his lock



opens the box, and removes the slip of paper from inside.

“heads”, signed Alice

He gives the slip to Alice.

Alice checks that Bob didn't cheat

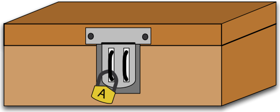
At this point, both Alice and Bob know the outcome of the coin toss.

Alice verifies that the slip of paper is one of the two that she prepared at the beginning, with her handwriting on it.

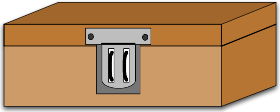
She sends her key to Bob.

Bob checks that Alice didn't cheat

Bob still has the other box.



He removes Alice's lock,



opens the box, and removes the slip of paper from inside.

$\boxed{\text{“tails”, signed Alice}}$

He checks that it contains the other coin value.

Locked Box Implementation

Commutative-key cryptosystems

Alice and Bob can carry out this protocol electronically using any *commutative-key* cryptosystem, that is, one in which $E_A \circ E_B = E_B \circ E_A$.[1]

RSA is commutative for keys A and B with a common modulus n, so we can use RSA in an unconventional way.

Rather than making the encryption exponent public and keeping the factorization of n private, we turn things around.

[1]Recall the related notion of "commutative cryptosystem" of lecture 11 in which the encryption and decryption functions for the *same* key commuted.

RSA as a commutative-key cryptosystem

Alice and Bob jointly chose primes p and q, and both compute $n = pq$.

Alice chooses an RSA key pair $A = ((e_A, n), (d_A, n))$, which she can do since she knows the factorization of n.

Similarly, Bob chooses an RSA key pair $B = ((e_B, n), (d_B, n))$ using the *same* n.

Alice and Bob both keep their key pairs private (until the end of the protocol, when they reveal them to each other to verify that there was no cheating).

Security remark

We note that this scheme may have completely different security properties from usual RSA.

In RSA, there are three different secrets involved with the key: the factorization of n, the encryption exponent e, and the decryption exponent d.

We have seen previously that knowing n and any two of these three pieces of information allows the third to be reconstructed.

Thus, knowing the factorization of n and e lets one compute d. It is also possible to factor n given both e and d.

The way RSA is usually used, only e is public, and it is believed to be hard to find the other two secrets.

A new use for RSA

Here we propose making the factorization of n public but keeping e and d private.

It may indeed be hard to find e and d, even knowing the factorization of n, but if it is, that fact is not going to follow from the difficulty of factoring n.

Of course, for security, we need more than just that it is hard to find e and d.

We also need it to be hard to find m given $c = m^e \bmod n$.

This is reminiscent of the discrete log problem, but of course n is not prime in this case.

Coin-flipping using commutative-key cryptosystems

We now implement the locked box protocol using RSA in this new way.

Here we assume that Alice and Bob initially know large primes p and q.

	Alice		Bob
1.	Choose RSA key pair A with modulus $n = pq$.		Choose RSA key pair B with modulus $n = pq$.
2.	Choose random $r \in \mathbf{Z}_{(n-1)/2}$. Let $m_i = 2r + i$, for $i \in \{0,1\}$. Let $c_i = E_A(m_i)$ for $i \in \{0,1\}$. Let $C = \{c_0, c_1\}$.	$\xrightarrow{C}$	Choose $c_a \in C$.
3.		$\xleftarrow{c_{ab}}$	Let $c_{ab} = E_B(c_a)$.
4.	Let $c_b = D_A(c_{ab})$.	$\xrightarrow{c_b}$	
5.		$\xleftarrow{B}$	Let $m = D_B(c_b)$. Let $i = m \bmod 2$. Let $r = (m - i)/2$. If $i = 0$ then "tails". If $i = 1$ then "heads".

	Alice	Bob
6.	Let $m = D_B(c_b)$. Check $m \in \{m_0, m_1\}$. If $m = m_0$ then "tails". If $m = m_1$ then "heads". $\xrightarrow{A}$	
7.		Let $c_a' = C - \{c_a\}$. Let $m' = D_A(c_a')$. Let $i' = m' \bmod 2$. Let $r' = (m' - i')/2$. Check $i' \neq i$ and $r' = r$.

Notes

In step (2), Alice chooses a random number r such that $r < (n-1)/2$.

This ensures that $m_0 = 2r$ and $m_1 = 2r+1$ are both in $\mathbf{Z}_n$.

Note that i and r can be efficiently recovered from m_i since i is just the low-order bit of m_i and $r = (m_i - i)/2$.

Correctness when Alice and Bob are honest

When both Alice and Bob are honest, Bob computes $c_{ab} = E_B(E_A(m_j))$ for some $j \in \{0, 1\}$.

In step 4, Alice computes c_b.
By the commutativity of E_A and E_B,

$$c_b = D_A(E_B(E_A(m_j))) = E_B(m_j).$$

Hence, in step 5, $m = m_j$ is one of Alice's strings from step 2.

A dishonest Bob

A dishonest Bob can control the outcome of the coin toss if he can find two keys B and B' such that $E_B(c_a) = E_{B'}(c'_a)$, where $C = \{c_a, c'_a\}$ is the set received from Alice in step 2.

In this case, $c_{ab} = E_B(E_A(m_j)) = E_{B'}(E_A(m_{1-j}))$ for some j. Then in step 4, $c_b = D_A(c_{ab}) = E_B(m_j) = E_{B'}(m_{1-j})$.

Hence, $m_j = D_B(c_b)$ and $m_{1-j} = D_{B'}(c_b)$, so Bob can obtain both of Alice's messages and then send B or B' in step 5 to force the outcome to be as he pleases.

To find such B and B', Bob would need to solve the equation

$$c_a^e \equiv {c'_a}^{e'} \pmod{n}$$

for e and e'. Not clear how to do this, even knowing the factorization of n.

Card dealing using locked boxes

The same locked box paradigm can be used for dealing a 5-card poker hand from a deck of cards.

Alice takes a deck of cards, places each card in a separate box, and locks each box with her lock.

She arranges the boxes in random order and ships them off to Bob.

Bob picks five boxes, locks each with his lock, and send them back.

Alice removes her locks from those five boxes and returns them to Bob.

Bob unlocks them and obtains the five cards of his poker hand.

Further details are left to the reader.

Oblivious Transfer

Generalization of coin-flipping protocol

In the locked box coin-flipping protocol, Alice has two messages m_0 and m_1.

Bob gets one of them.

Alice doesn't know which (until Bob tells her).

Bob can't cheat to get both messages.

Alice can't cheat to learn which message Bob got.

The *oblivious transfer problem* abstracts these properties from particular applications such as coin flipping and card dealing.

One-of-two oblivious transfer

In *one-of-two oblivious transfer* (OT^2_1), Alice has two secrets, s_0 and s_1.

Bob always gets exactly one of the secrets, each with probability $1/2$.

Alice does not know which one Bob gets.

The locked box protocol is one way to implement one-of-two oblivious transfer.

Another 1-of-2 OT protocol using blinding [2]

	Alice		Bob
1.	Secrets s_0 and s_1. Choose RSA key (n, e, d). Let $y_i = E_e(s_i)$, $i \in \{0, 1\}$.	$\xrightarrow{n,e,y_0,y_1}$	
2.		$\xleftarrow{c}$	Choose random $b \in \{0, 1\}$. Choose random $r \in \mathbf{Z}_n^*$. Compute $c = y_b E_e(r) \bmod n$.
3.	Let $c' = D_d(c) \equiv s_b r \pmod{n}$.	$\xrightarrow{c'}$	
4.			Output $c' r^{-1} \bmod n = s_b$.

[2] This protocol is adapted from notes by David Wagner, U.C. Berkeley, CS276, lecture 29, May 2006.

Analysis

- In step 1, Alice sends Bob encryptions of both secrets.
- In step 2, Bob chooses one of Alice's encryptions, blinds it, and returns the result to Alice.
- In step 3, Alice decrypts whatever Bob sends her, which allows Bob to unblind the decryption and recover the secret he chose in step 2.

Alice's other secret is safe assuming semi-honest parties (see lecture 21) as long as RSA is secure under a limited chosen ciphertext attack (since that is what Alice permits in step 3).

Bob's blinding prevents Alice from knowing which secret he learned.

Blinding

Blinding is a powerful concept in privacy-preserving protocols.

Bob computes a *blinding factor* B by encrypting a random $r \in \mathbf{Z}_n^*$, so $B = E_e(r) \bmod n$ is also a random element of $\mathbf{Z}_n^*$.

Alice gets no information about b from $c = y_b B \bmod n$ in step 2 since c is just another random element in $\mathbf{Z}_n^*$.

Because Alice is using RSA, when she decrypts c, the resulting c' is also random.

$$c' = D_d(c) \equiv c^d \equiv (y_b B)^d \equiv (s_b^e r^e)^d \equiv (s_b r)^{ed} \equiv s_b r \pmod{n}.$$

Bob removes the blinding factor by computing $s_b = c' r^{-1} \bmod n$.

The Millionaires' Problem

The Millionaires' Problem

The Millionaires' problem, introduced by Andy Yao in 1982, began the study of *privacy-preserving multiparty computation*.

Alice and Bob want to know who is the richer without revealing how much they are actually worth.

Alice is worth I million dollars; Bob is worth J million dollars.

They want to determine whether or not $I \geq J$, but at the end of the protocol, neither should have learned any more about the other person's wealth than is implied by the truth value of the predicate $I \geq J$.

Privacy-preserving multiparty computation

Another example is vote-counting.

Each voter has an input $v_i \in \{0, 1\}$ indicating their no/yes vote on an issue.

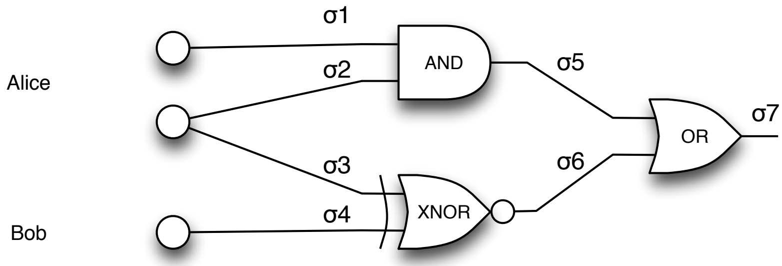
The goal is to collectively compute $\sum v_i$ while maintaining the privacy of the individual v_i.

Privacy-Preserving Boolean Function Evaluation

Boolean functions computed by circuits

Let $\bar{z} = f(\bar{x}, \bar{y})$, where $\bar{x}$, $\bar{y}$, and $\bar{z}$ are bit strings of lengths n_x, n_y, and n_z, respectively, and f is a Boolean function computed by a polynomial size Boolean circuit C_f with $n_x + n_y$ input wires and n_z output wires.

Example:



Private evaluation

In a *private evaluation* of C_f, Alice furnishes the (private) input data to the first n_x input wires, and Bob furnishes the (private) input data for the remaining n_y input wires. The n_z output wires should contain the result $\bar{z} = f(\bar{x}, \bar{y})$. The corresponding *functionality* is

$$F(\bar{x}, \bar{y}) = (\bar{z}, \bar{z}).$$

Alice and Bob should learn nothing about each others inputs or the intermediate values of the circuit, other than what is implied by their own inputs and the output values $\bar{z}$.

Circuit evaluation

An evaluation of a circuit assigns a Boolean value σ_w to each wire of the circuit. The input wires are assigned the corresponding input values.

Let G be a gate with input wires u and v and output wire w that computes the Boolean function $g(x, y)$. In a correct assignment, $\sigma_w = g(\sigma_u, \sigma_v)$.

A complete evaluation of the circuit first assigns values to the input wires and then works its way down the circuit, assigning a value to the output wire of any gate whose inputs have already received values.

Private circuit evaluation

In a *private circuit evaluation*,

- Both Alice and Bob learn the output values of the circuit;
- Neither Alice nor Bob gain any information about each others private input values except for whatever is implied by their own input values and the circuit output.

We present two different schemes for privately evaluating a circuit:

- Value shares;
- Garbled circuits.

Implementation Using Value Shares

Value shares

In a private evaluation using *value shares*, we split each value σ_w into two random shares a_w and b_w such that $\sigma_w = a_w \oplus b_w$.

- Alice knows a_w; Bob knows b_w.
- Neither share alone gives any information about σ_w, but together they allow σ_w to be computed.

After all shares have been computed for all wires, Alice and Bob exchange their shares a_w and b_w for each output wire w.

They are both then able to compute the circuit output.

Obtaining the shares

We now describe how Alice and Bob obtain their shares while maintaining the desired privacy.

There are three cases, depending on whether w is

1. An input wire controlled by Alice;
2. An input wire controlled by Bob;
3. The output wire of a gate G.

Alices input wires

1. Input wire controlled by Alice:

   Alice knows σ_w.

   She generates a random share $a_w \in \{0, 1\}$ for herself and sends Bob his share $b_w = a_w \oplus \sigma_w$.

Bobs input wires

2. Input wire controlled by Bob:

   Bob knows σ_w.

   Alice chooses a random share $a_w \in \{0, 1\}$ for herself.

   She prepares a table T:

σ	$T[\sigma]$
0	a_w
1	$a_w \oplus 1$.

   Bob requests $T[\sigma_w]$ from Alice via OT^2_1 and takes his share to be $b_w = T[\sigma_w] = a_w \oplus \sigma_w$.

Obtaining shares for gate output wires

3. Output wire of a gate G:
   Let G have input wires u, v and compute function $g(x, y)$.
   Alice chooses random share $a_w \in \{0, 1\}$ for herself.
   She computes the table

$$\begin{array}{rcl} T[0,0] & = & a_w \oplus g(a_u, a_v) \\ T[0,1] & = & a_w \oplus g(a_u, a_v \oplus 1) \\ T[1,0] & = & a_w \oplus g(a_u \oplus 1, a_v) \\ T[1,1] & = & a_w \oplus g(a_u \oplus 1, a_v \oplus 1) \end{array}$$

   (Equivalently, $T[r, s] = a_w \oplus g(a_u \oplus r, a_v \oplus s)$.)

   Bob requests $T[b_u, b_v]$ from Alice via OT^4_1 and takes his share to be $b_w = T[b_u, b_v] = a_w \oplus g(\sigma_u, \sigma_v)$.

Remarks

1. Alice and Bobs shares for w are both independent of σ_w.
   - Alices share is chosen uniformly at random.
   - Bobs share is always the XOR of Alices random bit a_w with something independent of a_w.
2. This protocol requires n_y executions of OT_1^2 to distribute the shares for Bobs inputs, and one OT_1^4 for each gate.[3]
3. This protocol assumes semi-honest parties.
4. This protocol generalizes readily from 2 to m parties.
5. Bob does not even need to know what function each gate G computes. He only uses his private inputs or shares to request the right line of the table in each of the several OT protocols.

[3] Note: The n_y executions of OT_1^2 can be eliminated by having Bob produce the shares for his input wires just as Alice does for hers. Our approach has the advantage of being more uniform since Alice is in charge of distributing the shares for all wires.

Implementation Using Garbled Circuits

Garbled circuits

A very different approach to private circuit evaluation is the use of *garbled circuits*.

The idea here is that Alice prepares a garbled circuit in which each wire has associated with it a tag corresponding to 0 and a tag corresponding to 1.

Associated with each gate is a template that allows the tag that represent the correct output value to be computed from the tags representing the input values.

This is all done in a way that keeps hidden the actual values that the tags represent.

A sketch of the protocol

After creating the circuit, Alice, who knows all of the tags, uses OT_1^2 to send Bob the tags corresponding to values on the input wires that he controls.

She also sends him the tags corresponding to the values on the input wires that she controls.

Bob then evaluates the circuit all by himself, computing the output tag for each gate from the tags on the input wires.

Finishing up

At the end, he knows the tags corresponding to the output wires.

Alice knows which Boolean values those tags represent, which she sends to Bob (either before or after he has evaluated the circuit).

In this way, Bob learns the output of the circuit, which he then sends to Alice.

Role of the tags

The scrambled gate is a 4-line table giving the output tag corresponding to each of the possible 4 input values.

Each line of the table is encrypted differently.

The input tags to the gate allow the corresponding table item to be decrypted.

Evaluating the circuit then amounts to decrypting ones way though the circuit, gate by gate, until getting the output tag.

Remarks

1. The OT_1^2 protocol steps used to distribute the tags for the wires that Bob controls keeps his inputs private from Alice. The privacy of Alices inputs and intermediate circuit values from Bob relies on the encryption function used to hide the association between tags and values.
2. The security of the protocol relies on properties of the encryption function that we have not stated.
3. This protocol requires only n_y executions of OT_1^2 and hence should be considerably faster to implement than the share-based protocol.
4. This protocol also assumes semi-honest parties.
5. Doesnt easily generalize to more than two parties.
6. Bob doesnt need to know the function each gate computes. He only needs the associated templates.

Appendix: Problems at Least as Hard as Factoring

Factoring n of the form pq

We've seen several protocols involving computation over $\mathbf{Z}_n^*$, where n is the product of two distinct odd primes p and q.

Associated with these protocols are other problems can be shown to be at least as hard as factoring, since a feasible solution to any of them could be used to factor n.

We give two examples.

Finding square roots

Theorem

Let $n = pq$ with p, q distinct odd primes. Let $y \in \mathrm{QR}_n$, and let $\{\pm u, \pm v\}$ be the four distinct square roots of $y \pmod{n}$. Given n, u, v, one can factor n in polynomial time.

Proof.

We have $u^2 \equiv v^2 \equiv y \pmod{n}$, so $n \mid (u^2 - v^2) = (u - v)(u + v)$.

Since $u \not\equiv \pm v \pmod{n}$, then n does not divide $(u - v)$ and n does not divide $(u + v)$. But n does divide $(u - v)(u + v)$, so $d = \gcd(u - v)$ is a proper divisor of n.

Hence, $\{d, n/d\} = \{p, q\}$. $\square$

Finding RSA decryption exponent knowing public key

Theorem (Boneh[4])

Given RSA parameters n, e, d, one can factor n in polynomial time.

Proof.

1. $n = pq$ for p, q distinct odd primes.
2. $\phi(n) = (p-1)(q-1)$, so $\phi(n)$ is even.
3. $ed \equiv 1 \pmod{\phi(n)}$, so $\phi(n) \mid k$, where $k = ed - 1$.
4. Can write $k = 2^t r$ with $t \geq 1$ and r odd.
5. $g^k \equiv 1 \pmod{n}$ for every $g \in \mathbf{Z}_n^*$.
6. Compute sequence $g^{k/2}, g^{k/4}, \ldots, g^{k/2^t}$. For 1/2 of the $g \in \mathbf{Z}_n^*$, sequence contains $x \in \sqrt{1}$, where $x \notin \{1, -1\}$.
7. Then $\gcd(x-1, n) \in \{p, q\}$.

$\square$

[4] Dan Boneh, "Twenty Years of Attacks on the RSA Cryptosystem", Notices of the AMS 46, 2 (1999), 203–213.