CPSC 467: Cryptography and Security

Michael J. Fischer

Lecture 22
November 17, 2020

Locked Boxes

Oblivious Transfer

Homomorphic Encryption

Fully Homomorphic Encryption

Locked Boxes

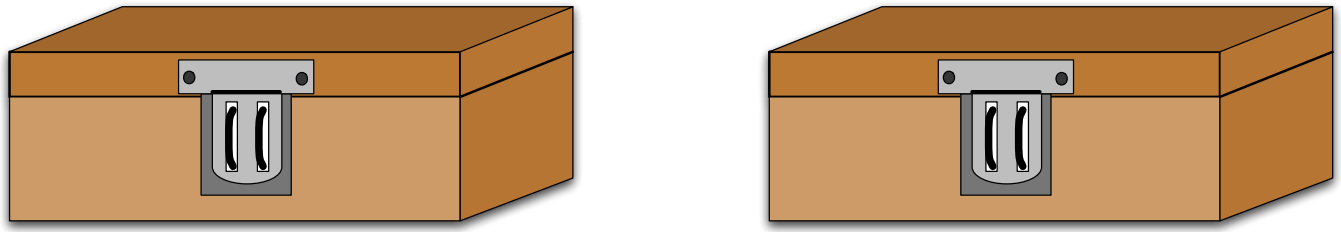
Locked boxes

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.

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.



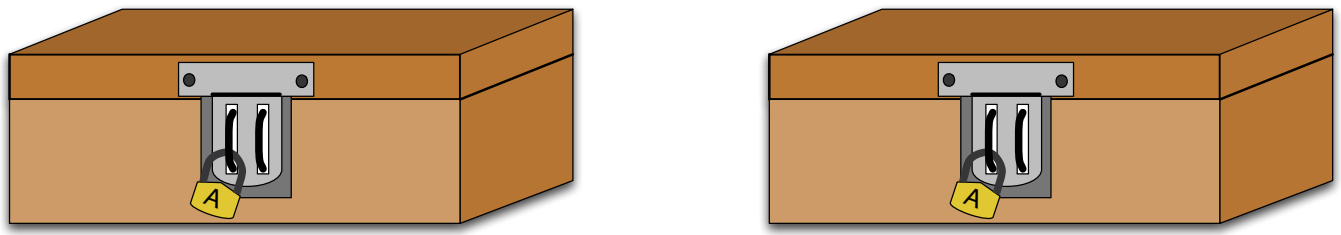
Alice writes "heads" on a slip of paper and "tails" on another.

"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.

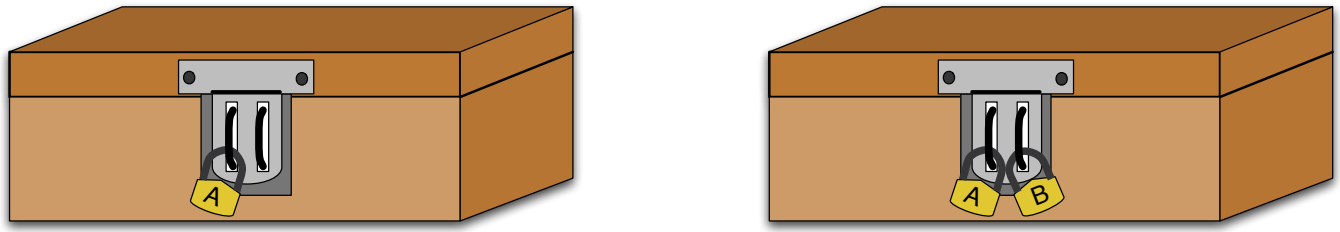


She then gives both locked boxes to Bob, in some random order.

Bob adds his lock

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

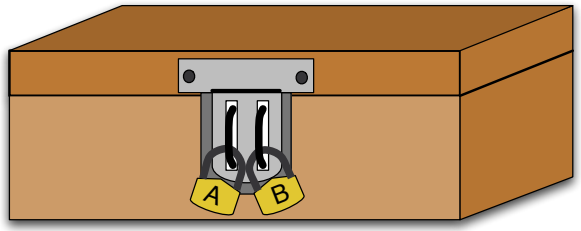
He chooses one of the boxes and locks it with his own padlock, for which he has the only key.



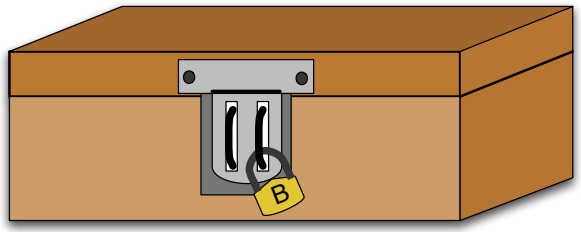
He gives the doubly-locked box back to Alice.

Alice removes her lock

Alice gets



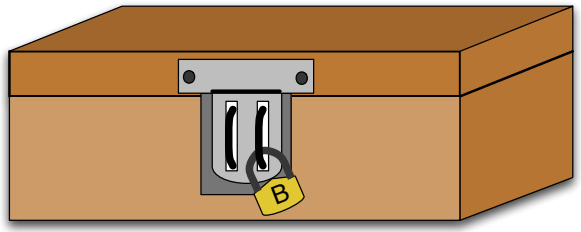
She removes her lock.



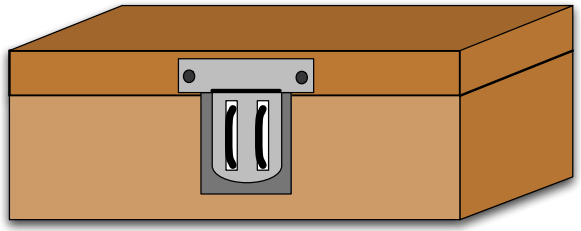
and returns the box to Bob.

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

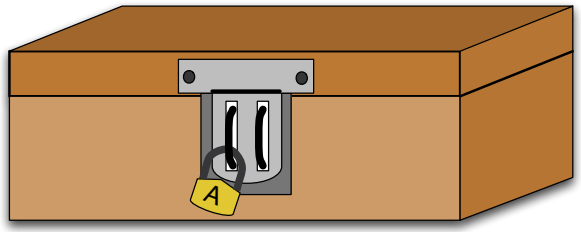
At this point, both Alice and Bob know the outcome of the coin toss.

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

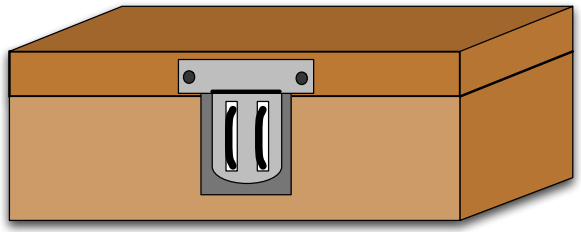
She sends the key to her locks to Bob.

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.

"tails", signed Alice

He checks that it contains the other coin value.

What has been accomplished?

At the end of the locked box coin-flipping protocol, Alice knows that Bob got the message in a box of his choosing, but she does not know which box he chose until he opens it and send her the paper inside.

Bob on the other hand does not know the contents of either box before he makes his choice.

For Alice to know that Bob didn't cheat, she has to be able to check the slip of paper that Bob sends back to her to make sure it is the same as she put into one of the boxes at the beginning.

Implementation

Turning the locked box protocol into an actual implementation requires some sort of a commutative scheme for implementing the locks.

One possibility is for Alice and Bob to have random *blinding factors*, which they have each committed to.

Putting a lock on a box or removing it consists of simply XORing the box with the blinding factor.

Because blinding factors are random, after blinding by Alice, the boxes are indistinguishable to Bob.

Similarly, after Bob puts his lock on the chosen box, Alice can't tell which of the original boxes he has sent back.

Card dealing using locked boxes

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

1. Alice takes a deck of cards, places each card in a separate box, and locks each box with her lock.
2. She arranges the boxes in random order and ships them off to Bob.
3. Bob picks five boxes, locks each with his lock, and send them back to Alice.
4. Alice removes her locks from those five boxes and returns them to Bob.
5. 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_1^2), Alice has two secrets, s_0 and s_1.

Bob 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.

Strengthening one-of-two oblivious transfer

A slightly stronger version of OT_1^2, useful in secure multiparty computation, lets Bob choose which of Alice's secrets he gets, but his choice is hidden from Alice.

Upon completion, Bob has the secret he requested, but Alice has no idea which one he got.

This is just a brief overview of some of the techniques that are being used in advanced cryptographic protocols.

Homomorphic Encryption

Goals of encryption

The main goal of encryption is to provide data confidentiality.

Normally, there is a lot you can do with your unencrypted data: analyze, search, compute, etc.

However, once data is encrypted there is not much you can do with it.

Encrypted data $\rightarrow$ secured and useless
Unencrypted data $\rightarrow$ unsecured and useful

Working with encrypted data

Solution: Encrypt – decrypt – perform operations – re-encrypt

Problems: Can get very expensive very quickly. Privacy issues.

Another solution: Perform at least *some* operations on encrypted data *without* decrypting it.

Problems: How do we do that? What operations should be allowed? Will it affect security properties of the encryption scheme?

Homomorphic encryption

Informally, homomorphic encryption is an encryption scheme with a special property that allows operations applied to ciphertext to be carried over to the plaintext.

Group homomorphism

In mathematics, a *group* is a set equipped with a binary operation that combines any two elements to form a third element in such a way that four conditions called group axioms are satisfied, namely closure, associativity, identity and invertibility.[1]

A *group homomorphism* from $(G, *)$ to $(H, \cdot)$ is a function $h : G \to H$ such that $\forall u, v \in G$ it holds that $h(u * v) = h(u) \cdot h(v)$.

[1] From Wikipedia https://en.wikipedia.org/wiki/Group_(mathematics).

Homomorphic encryption

Let $\mathcal{M}$ be the set of plaintext messages, $\mathcal{C}$ be the set of ciphertext messages, and $\mathcal{K}$ be the set of keys.

An encryption scheme is *homomorphic* if for any given encryption key $k \in \mathcal{K}$ the encryption function E_k satisfies:

$$\forall m_1, m_2 \in \mathcal{M},\ E(m_1 \odot_{\mathcal{M}} m_2) = E(m_1) \odot_{\mathcal{C}} E(m_2)$$

for some operators $\odot_{\mathcal{M}}$ in $\mathcal{M}$ and $\odot_{\mathcal{C}}$ in $\mathcal{C}$.

We want $(\mathcal{M}, \odot_{\mathcal{M}})$ and $(\mathcal{C}, \odot_{\mathcal{C}})$ to form a group homomorphism under E_k.

C. Fontaine and F. Galand, *A Survey of Homomorphic Encryption for Nonspecialists*, EURASIP Journal on Information Security, 2007

Types of homomorphism

An encryption scheme can be homomorphic with respect to one or more group operators.

An encryption scheme is *additively* homomorphic if we consider the addition operator, and *multiplicatively* homomorphic if we consider the multiplication operator.

Types of homomorphic encryption

Partially homomorphic encryption – it is possible to perform operations on encrypted data with respect to one group operator. For example, from $E(x)$, $E(y)$ compute $E(x + y)$ but not $E(x * y)$.

Fully homomorphic encryption – it is possible to perform operations on encrypted data with respect to two group operator. For example, from $E(x)$, $E(y)$ compute $E(x + y)$ and $E(x * y)$.

Somewhat homomorphic encryption – it is possible to perform a limited number of operations on encrypted data with respect to two group operators. For example, we can only evaluate low-degree polynomials over encrypted data.

Applications of homomorphic encryption

- Cloud computing (untrusted third parties can be used)
- E-voting (votes can be counted without revealing what they are)
- Private information retrieval (searching encrypted databases)

Partially homomorphic encryption schemes

There are many encryption schemes which have the desired homomorphic property.

You should be familiar with at least some of them:

- RSA (multiplicatively)
- ElGamal (multiplicatively)
- Exponential Elgamal (additively for small numbers)
- Goldwasser-Micali (additively)

(Plain) RSA

Public key: (e, N)
Private key: (d, N)
Encryption function: $E(m) = m^e \bmod N$

Multiplicatively homomorphic property:

$$\begin{aligned} E(m_1) * E(m_2) &= (m_1^e \bmod N) * (m_2^e \bmod N) \bmod N \\ &= m_1^e * m_2^e \bmod N \\ &= (m_1 * m_2)^e \bmod N \\ &= E(m_1 * m_2) \end{aligned}$$

ElGamal

Public key: (p, g, b), where $b = g^x$
Private key: (x)
Encryption function: $E(m) = (g^r, m * b^r)$ for a random $r \in Z_{\phi(p)}$

Multiplicatively homomorphic property:

$$\begin{aligned} E(m_1) * E(m_2) = \\ (g^{r_1}, m_1 * b^{r_1})(g^{r_2}, m_2 * b^{r_2}) = \\ (g^{r_1+r_2}, (m_1 * m_2)b^{r_1+r_2}) = \\ E(m_1 * m_2) \end{aligned}$$

Exponential ElGamal

Public key: (p, g, b), where $b = g^x$
Private key: (x)
Encryption function: $E(m) = (g^r, g^m * b^r)$ for a random $r \in Z_{\phi(p)}$

Additively homomorphic property:

$$E(m_1) * E(m_2) =$$
$$(g^{r_1}, g^{m_1} * b^{r_1})(g^{r_2}, g^{m_2} * b^{r_2}) =$$
$$(g^{r_1+r_2}, g^{(m_1*m_2)} b^{r_1+r_2}) =$$
$$E(m_1 + m_2)$$

Q) Why does this scheme work only for small messages m?
A) Decrypting requires finding the discrete log of g^m.

Goldwasser-Micali

Public key: (y, N), where $y \in Q_N^{00}$
Private key: (p, q), where $N = p * q$
Encryption function: $E(b) = r^2 y^b \bmod N$, where b is one bit of plaintext and $r \in Z^*_{\phi(N)}$

Decryption function: If $E(b) \in Q_N^{11}$, then $b = 0$, otherwise $b = 1$.

Additively homomorphic property:

$$\begin{aligned} E(b_1) * E(b_2) &= \\ r_1^2 y^{b_1} r_2^2 y^{b_2} &= \\ (r_1 r_2)^2 y^{b_1 + b_2} &= \\ E(b_1 \oplus b_2) & \end{aligned}$$

Security notions for encryption schemes

Combination of security goals and attack scenarios.[2]
Goals:

- Indistinguishability (IND). The attacker does not learn anything about plaintext x from ciphertext y.
- Non-malleability (NM). Based on ciphertext y the attacker cannot produce y' so that the corresponding plaintexts x and x' are meaningfully related.

Attack scenarios:

- Chosen–plaintext attack (CPA).
- Non–adaptive chosen–ciphertext attack (CCA1).
- Adaptive chosen–ciphertext attack (CCA2).

[2] See Wikipedia Ciphertext indistinguishability.

Security of homomorphic encryption

Let's (informally) rephrase what homomorphic encryption is.

"If you encrypt some plaintext using homomorphic encryption, then by changing the ciphertext you can change the corresponding plaintext in predictable ways".

Q: Is it a good or bad property?

Security of homomorphic encryption (cont.)

Non–malleability is a desirable security goal for encryption schemes so that the attacker cannot tamper with the ciphertext to affect the plaintext and go undetected.

However, homomorphic encryption implies malleability!

To reconcile this situation, we want an encryption scheme to be non-malleable except for some desired operations.

However, it's difficult to capture the notion of "some malleability allowed."[3]

[3] B. Hemenway and R. Ostrovsky, *On Homomorphic Encryption and Chosen-Ciphertext Security*, PKC 2012.

Fully Homomorphic Encryption

First FHE scheme

The first fully homomorphic encryption scheme using lattice–based cryptography was presented by Craig Gentry in 2009.[4]

Later in 2009 a second fully homomorphic encryption scheme which does not require ideal lattices was presented.[5]

A lot has changed since then!

[4] C. Gentry, *Fully Homomorphic Encryption Using Ideal Lattices*, STOC 2009.

[5] M. van Dijk, C. Gentry, S. Halevi and V. Vaikuntanathan, *Fully Homomorphic Encryption over the Integers*, Eurocrypt 2010.

FHE performance

Gentry estimated[6] that performing a Google search with encrypted keywords would increase the amount of computing time by about a trillion. Moore's law calculates that it would be 40 years before that homomorphic search would be as efficient as a search today.

At Eurocrypt 2010, Craig Gentry and Shai Halevi presented a working implementation of fully homomorphic encryption.
Martin van Dijk about the efficiency:

"Computation, ciphertext-expansion are polynomial, but a rather large one..."

[6] *IBM Touts Encryption Innovation. New technology performs calculations on encrypted data without decrypting it*, computerworld.com, M. Cooney.

Current FHE efforts

FHE has been a very popular research area over the past decade.

See Wikipedia for an overview of progress made in terms of better algorithms, greater efficiency, available implementations, and standardization.