Today: Privacy with utility

Recall: We had a module on platform security, where we showed how to completely hide a message using encryption.

Today: What if we want to hide the message but at the same time leak some specific information, for the sake of utility?

Things become much more complicated...

Preview to advanced topics in cryptography (some which are covered in 6.875)

Examples:

1. Encrypted email

Suppose we want to hide the content of the emails that we recieve.

Why would we want to hide this data?

1. Data gets stolen.
2. Data gets misused or abused.
3. Data gets mispurposed in ways that we don't expect.

Lets hide our emails using a public-key encryption scheme!

If we use a secret key scheme we will need to preshare a secret key with everyone who sends us an email...

Hiding the content of our emails comes at a big price!

Ex. We cannot use spam filters!

Goal: Encrypt our emails but leak only whether it is a spam or not.

The spam filter is a function F that takes as input a message M and outputs 0 or 1, corresponding to spam or not spam.

Idea: Use a special encryption scheme that given sk and F,

allows to generate a special secret-key sk(F) such that:

Given Enc(sk,M), sk(F), one can efficiently compute F(m), but learn nothing more about M!

This is called a functional encryption scheme.

Constructing a functional encryption scheme is quite difficult!

Known constructions are highly inefficient.

Constructing a functional encryption scheme that allows to leak a *single* functional key sk(F) has been constructed in 2013 from LWE (Learning with Error), which is a standard cryptographic assumption

Constructing a functional encryption scheme that allows to leak many functional secret keys seems to be significantly more challanging and is equivalent to indistinguishable obfuscation.

2. Obfuscation: The goal is to publish a program that leaks nothing except the input/output behavior of the program.

This is one of the most important goals in cryptography today!

One can use program obfuscation to construct almost any cryptographic primitive we can think of, and more!

Example: Software updates.

Suppose a software company found a bug in their software and wants to post an update, but does not wish to reveal the bug!

Unfortunately, constructing (and even defining) the notion of obfuscation is very tricky!

There has been a recent breakthrough showing how to obtain "indistinguishability obfuscation".

This is extremely useful for cryptography!!!

Unfortunately, is extremely inefficient.

Both functional encryption and obfuscation seem far from practical.

There are a few examples of privacy with utility that seem to be easier,

and are used in practice.

3. Zero-knowledge proofs:

Suppose I want to prove to you the validity of a statement

without revealing any information about why the statement is true?

This is called a zero-knowledge proof (ZKP),

and is one of the magical concepts of cryptography, invented in the mid 80s.

The reason it is so magical is that at first sight, it is clearly impossible!

The reason is that a proof is information!

For example, I can use it to convince others that the statement is true.

As we said throughout this course,

cryptography is the art of making the impossible possible!

Zero-knowledge is made possible by changing the classical model of a proof,

to allow the proofs to be interactive and randomized!

It is remarkable that even though proofs were around for thousands of years,

they were studied by ancient greeks, starting from Euclid, and were developed into an active

area in mathematics, called proof theory, throughout these years, proofs always consisted of a

of a single document, that could be deemed to be either "valid" or "invalid".

This changed with the introduction of zero-knowledge proofs!

Interactive Proofs:

An interactive proof is an interactive and probabilistic protocol between a prover and an efficient verifier.

If the statement is true, then an honest prover can convince the verifier to accept with probability 1.

If the statement is false, any malicious prover can convince the verifier to accept only with small probability.

"Theorem": Any proof can be converted into a zero-knowledge interactive proof.

Zero-knowledge proofs are used in practice.

They are used for authentication: A user proves that he "knows" a secret key using a ZK proof.

They are used on the block-chain to shield the transactions, which are known to reveal private information.

4. Secure multi-party computation (MPC)

Allows a set of entities to jointly do a computation on their sensitive data, and learn only the output,

without revealing anything more about their sensitive data.

Again, the goal here is to hide the sensitive data, but at the same time to learn the output.

Example: This can allow hospitals to do joint computations on their sensitive patient data.

Next class, we will talk about what to do if the output you want to learn contains sensitive information.

Starting from the mid 80's we know how to compute any multi-party function securely.

This uses very interesting tools (which we did not discuss in class), such as secret sharing,

oblivious transfer, commitments, and also ZK proofs.