CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Lecture 6
September 16, 2013

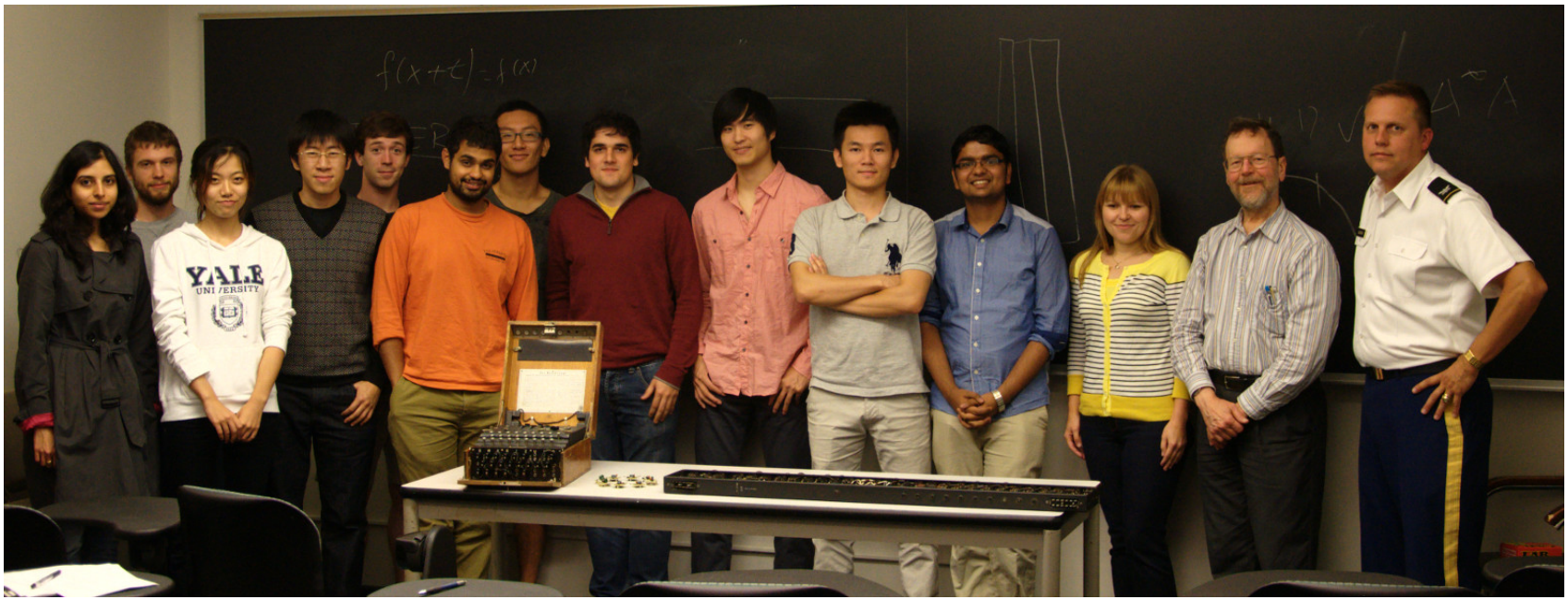
Enigma machine exhibit

Enigma exhibit

Col. Robert W. Sadowski from the U. S. Military Academy at West Point provided the class with the rare opportunity to see a real German Enigma machine up close and to hear about its use during World War II.

We give our heartfelt thanks to Col. Sadowski for his fascinating presentation and for bringing the machine for us to see.

Enigma exhibit day



Left to right: CPSC 467/567 students, TA Ewa Syta, Prof. Michael Fischer, Col. Robert Sadowski.
On table, left to right: Enigma machine, dynamic shift register demo, vacuum tube register from Eniac computer.

Using block ciphers

Block ciphers

Recall: *Block ciphers* map b-bit plaintext blocks to b-bit ciphertext blocks.

Block ciphers typically operate on fairly long blocks, e.g., 64-bits for DES, 128-bits for Rijndael (AES).

Block ciphers can be designed to resist known-plaintext attacks, provided b is large enough, so they can be pretty secure, even if the same key is used to encrypt a succession of blocks, as is often the case.

What goes wrong if b is small?

Using a block cipher

One rarely wants to send messages of exactly the block length.

To use a block cipher to encrypt arbitrary-length messages:

- Divide the message into blocks of size b.
- Pad the last partial block according to a suitable *padding rule* and/or add another block at the end.
- Use the block cipher in some *chaining mode* to encrypt the resulting sequence of blocks.

Padding

Padding extends the message to satisfy two requirements:

- The length must be a multiple of b.
- It must be possible to recover the exact original message from the padded message.

Just sticking 0's on the end of a message until its length is a multiple of b will not satisfy the second requirement.

A padding rule must describe how much padding was added.

Suggestions?

Padding rules

Here's one rule that works.

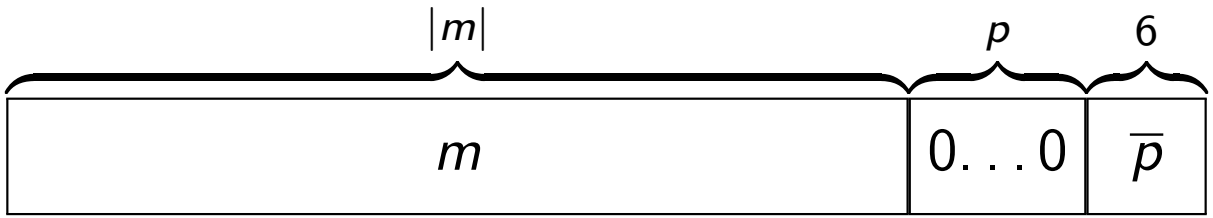
- Choose $\ell = \lceil \log_2 b \rceil$. This is the number of bits needed to represent (in binary) any number in the interval $[0 \ldots (b-1)]$.
- Choose p as small as possible so that $|m| + p + \ell$ is a multiple of b.
- Pad each message with p 0's followed by a length ℓ binary representation of p.

To unpad, interpret the last ℓ bits of the message as a binary number p; then discard a total of $p + \ell$ bits from the right end of the message.

Padding example

Example, $b = 64$:

- At most 63 0's ever need to be added, so a 6-bit length field is sufficient.
- A message m is then padded to become $m' = m \cdot 0^p \cdot \overline{p}$, where $\overline{p}$ is the 6-bit binary representation of p.
- p is chosen as small as possible so that $|m'| = |m| + p + 6$ is a multiple of 64.



Possible information leakage from padding

Suppose Alice uses AES to send 129-bit messages.

Eve has a plaintext-ciphertext pair (m', c') and intercepts a new cipher text c for an unknown message m.

Because of padding, both c and c' are two blocks long. Let c_2 and c_2' be the second blocks of each, respectively.

Then the last bit of m is the same as the last bit of m' iff $c_2 = c_2'$, so Eve learns the last bit of m.

Note: This only applies when AES is being used in ECB mode.

Symmetric cryptosystem families

Symmetric cryptosystem families

Symmetric (one-key) cryptosystems fall into two broad classes, *block ciphers* and *stream ciphers*.

- A block cipher encrypts large blocks of data at a time.
- A stream cipher encrypts a character stream in an on-line fashion, encrypting and outputting each byte before reading the next.

DES and AES are block ciphers. We now consider how to construct stream ciphers, either from scratch or based on block ciphers.

Structure of stream cipher

A stream cipher can be built from two components:

1. a cipher that is used to encrypt a given character;
2. a keystream generator that produces a different key to be used for each successive letter.

A commonly-used cipher is the simple XOR cryptosystem, also used in the one-time pad.

Rather than using a long random string for the keystream, we instead use a pseudorandom keystream generated on the fly using a state machine.

Like a one-time pad, a different master key (seed) must be used for each message; otherwise the system is easily broken.