CPSC 467b: Cryptography and Computer Security

Michael J. Fischer

Lecture 6
January 31, 2013

Enigma machine exhibit

Building stream cipher from PRSG

Building stream cipher from block cipher

- Byte padding
- Chaining modes

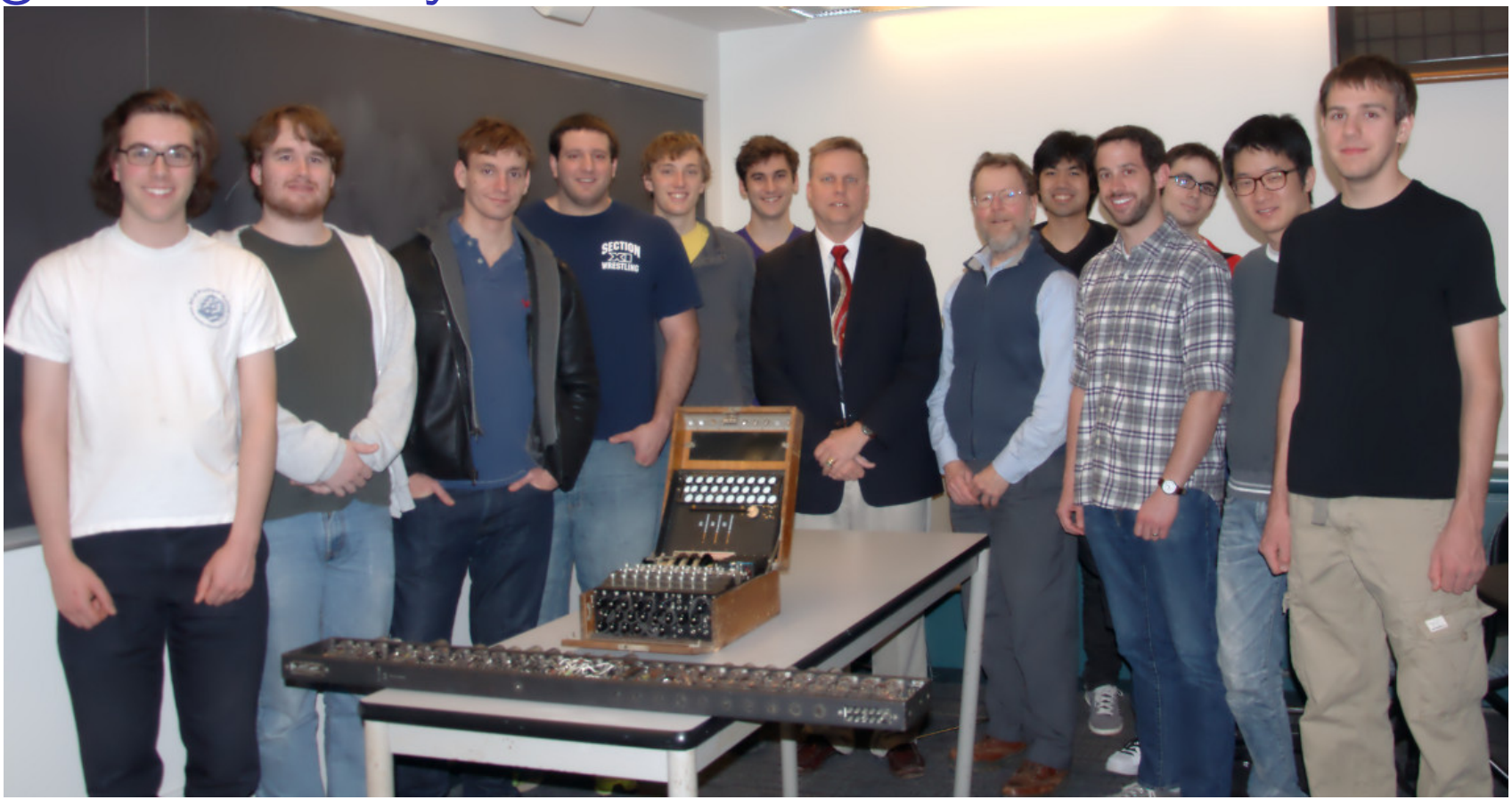
Enigma machine exhibit

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



Col. Robert Sadowski (center left), Prof. Michael Fischer (center right), and CPSC 467/567 class surrounding Enigma machine (center) and register from Eniac computer (foreground).

Building stream cipher from PRSG

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.

Pseudorandom sequence generator (PRSG)

A *pseudoramdom sequence generator (PRSG)* consists of:

1. a *seed* (or *master key*),
2. a *state*,
3. a *next-state generator*,
4. an *output function*.

The initial state is derived from the seed.

At each stage, the state is updated and the output function is applied to the state to obtain the next component of the output stream.

Security requirements

- The output of the PRSG must "look" random.
- Any regularities in the output of the PRSG give an attacker information about the plaintext.
- A known plaintext-ciphertext pair (m, c) gives the attacker a sample output sequence from the PRSG (namely, $m \oplus c$.)
- If the attacker is able to figure out the internal state, then she will be able to predict all future outputs of the generator and decipher the remainder of the ciphertext.

A pseudorandom sequence generator that resists all feasible attempts to predict future outputs, even knowing a sequence of past outputs, is said to be *cryptographically strong*.

Cryptographically strong PRSGs

Commonly-used linear congruential pseudorandom number generators typically found in software libraries are quite insecure.

After observing a relatively short sequence of outputs, one can solve for the state and correctly predict all future outputs.

Notes:

- The Linux `random()` is non-linear and hence much better, though still not cryptographically strong.
- We will return to pseudorandom number generation later in this course.
- See Goldwasser & Bellare Chapter 3 for an in-depth discussion of this topic.

Ideas for improving stream ciphers

As with one-time pads, the same keystream must not be used more than once.

A possible improvement: Make the next state depend on the current plaintext or ciphertext characters.

Then the generated keystreams will diverge on different messages, even if the key is the same.

Serious drawback: One bad ciphertext character will render the rest of the message undecipherable.

Building stream cipher from block cipher

Recall: Difference between block and stream ciphers

A block cipher cannot be used directly as a stream cipher.

- A stream cipher must output the current ciphertext byte before reading the next plaintext byte.
- A block cipher waits to output the current ciphertext block until a block's worth of message bytes have been accumulated.

We first return to the problem of using a block cipher to encrypt a sequence of blocks in an on-line fashion and then extend those ideas to become a true stream cipher.

Padding revisited

Lecture 4 presented a method of *bit padding* to turn an abtirary bit string into one whose length is a multiple of the block length.

Often the underlying message consists of a sequence of bytes, and a block comprises some number b of bytes.

This enables *byte padding* methods to be used, some of which are very simple.

PKCS7 padding

PKCS7 #7 is a message syntax described in internet RFC 2315. It's padding rule is to fill a partially filled last block having k "holes" with k bytes, each having the value k when regarded as a binary number.

For example, if the last block is 3 bytes short of being full, then the last 3 bytes are set to the values 03 03 03.

On decoding, if the last block of the message does not have this form, then a decoding error is indicated.

Example: The last block cannot validly end in . . . 25 00 03.

Chaining mode

A *chaining mode* tells how to encrypt a sequence of plaintext blocks $m_1, m_2, \ldots, m_t$ to produce a corresponding sequence of ciphertext blocks $c_1, c_2, \ldots, c_t$, and conversely, how to recover the m_i's given the c_i's.

Electronic Codebook Mode (ECB)

Each block is encrypted separately.

- To encrypt, Alice computes $c_i = E_k(m_i)$ for each i.
- To decrypt, Bob computes $m_i = D_k(c_i)$ for each i.

This is in effect a monoalphabetic cipher, where the "alphabet" is the set of all possible blocks and the permutation is E_k.

Cipher Block Chaining Mode (CBC)

Prevents identical plaintext blocks from having identical ciphertexts.

- To encrypt, Alice applies E_k to the XOR of the current plaintext block with the previous ciphertext block. That is, $c_i = E_k(m_i \oplus c_{i-1})$.
- To decrypt, Bob computes $m_i = D_k(c_i) \oplus c_{i-1}$.

To get started, we take $c_0 = \mathrm{IV}$, where IV is a fixed *initialization vector* which we assume is publicly known.

Output Feedback Mode (OFB)

Similar to a one-time pad, but keystream is generated from E_k.

- To encrypt, Alice repeatedly applies the encryption function to an *initial vector (IV)* k_0 to produce a stream of *block keys* $k_1, k_2, \ldots$, where $k_i = E_k(k_{i-1})$.

  The block keys are XORed with successive plaintext blocks. That is, $c_i = m_i \oplus k_i$.
- To decrypt, Bob applies exactly the same method to the ciphertext to get the plaintext.
  That is, $m_i = c_i \oplus k_i$, where $k_i = E_k(k_{i-1})$ and $k_0 = IV$.

Cipher-Feedback Mode (CFB)

Similar to OFB, but keystream depends on previous messages as well as on E_k.

- To encrypt, Alice computes the XOR of the current plaintext block with the encryption of the previous ciphertext block. That is, $c_i = m_i \oplus E_k(c_{i-1})$.
  Again, c_0 is a fixed initialization vector.
- To decrypt, Bob computes $m_i = c_i \oplus E_k(c_{i-1})$.

Note that Bob is able to decrypt without using the block decryption function D_k. In fact, it is not even necessary for E_k to be a one-to-one function (but using a non one-to-one function might weaken security).

OFB, CFB, and stream ciphers

Both CFB and OFB are closely related to stream ciphers.
In both cases, c_i is m_i XORed with some function of data that came before stage i.

Like a one-time pad, OFB is insecure if the same key is ever reused, for the sequence of k_i's generated will be the same.
If m and m' are encrypted using the same key k, then $m \oplus m' = c \oplus c'$.

CFB avoids this problem, for even if the same key k is used for two different message sequences m_i and m'_i, it is only true that $m_i \oplus m'_i = c_i \oplus c'_i \oplus E_k(c_{i-1}) \oplus E_k(c'_{i-1})$, and the dependency on k does not drop out.

Propagating Cipher-Block Chaining Mode (PCBC)

Here is a more complicated chaining rule that nonetheless can be deciphered.

- To encrypt, Alice XORs the current plaintext block, previous plaintext block, and previous ciphertext block.
  That is, $c_i = E_k(m_i \oplus m_{i-1} \oplus c_{i-1})$. Here, both m_0 and c_0 are fixed initialization vectors.
- To decrypt, Bob computes $m_i = D_k(c_i) \oplus m_{i-1} \oplus c_{i-1}$.

Recovery from data corruption

In real applications, a ciphertext block might be damaged or lost. An interesting property is how much plaintext is lost as a result.

- With ECB and OFB, if Bob receives a bad block c_i, then he cannot recover the corresponding m_i, but all good ciphertext blocks can be decrypted.
- With CBC and CFB, Bob needs both good c_i and c_{i-1} blocks in order to decrypt m_i. Therefore, a bad block c_i renders both m_i and m_{i+1} unreadable.
- With PCBC, bad block c_i renders m_j unreadable for all $j \geq i$.

Error-correcting codes applied to the ciphertext may be better solutions in practice since they minimize lost data and give better indications of when irrecoverable data loss has occurred.

Other modes

Other modes can easily be invented.

In all cases, c_i is computed by some expression (which may depend on i) built from $E_k()$ and $\oplus$ applied to available information:

- ciphertext blocks $c_1, \ldots, c_{i-1}$,
- message blocks $m_1, \ldots, m_i$,
- any initialization vectors.

Any such equation that can be "solved" for m_i (by possibly using $D_k()$ to invert $E_k()$) is a suitable chaining mode in the sense that Alice can produce the ciphertext and Bob can decrypt it.

Of course, the resulting security properties depend heavily on the particular expression chosen.