CSE 410 Fall 2025
Privacy-Enhancing Technologies

Marina Blanton

Department of Computer Science and Engineering
University at Buffalo

Lecture 4: Protecting Data at Rest II
Symmetric Encryption

Recap

We are looking into realizing secure and efficient encryption

This is realized by means of a block cipher

The current encryption standard is AES

Advanced Encryption Standard (AES)

In 1997 NIST made a call for an unclassified publicly disclosed encryption algorithm available worldwide and royalty-free

- the goal was to replace DES with a new standard called AES
- the algorithm must be a symmetric block cipher
- the algorithm must support (at a minimum) 128-bit blocks and key sizes of 128, 192, and 256 bits

The evaluation criteria were:

- security
- speed and memory requirements
- algorithm and implementation characteristics

AES

In 1998 15 candidate AES algorithms were announced

They were narrowed to 5 in 1999: MARS, RC6, Rijndael, Serpent, and Twofish

- all five were thought to be secure

In 2001 Rijndael was adopted as the AES standard

- invented by Belgian researchers Deamen and Rijmen
- designed to be simple and efficient in both hardware and software on a wide range of platforms
- supports different block sizes (128, 192, and 256 bits)
- supports keys of different length (128, 192, and 256 bits)
- uses a variable number of rounds (10, 12, or 14)

Rijndael Design

Rijndael doesn't have a Feistel structure

- 2 out of 5 AES candidates (including Rijndael) don't use Feistel structure
- they process the entire block in parallel during each round

The operations are (3 substitution and 1 permutation operations):

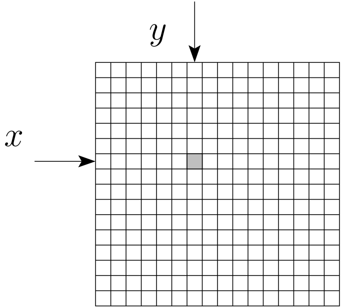
- SubBytes: byte-by-byte substitution using an S-box
- ShiftRows: a simple permutation
- MixColumns: a substitution using mod 2^8 arithmetics
- AddRoundKey: a simple XOR of the current state with a portion of the expanded key

Rijndael Design

- AddRoundKey is the only operation that uses key
  - that's why it is applied at the beginning and at the end
- all operations are reversible
- the decryption algorithm uses the expanded key in the reverse order
- the decryption algorithm, however, is not identical to the encryption algorithm

SubBytes

- The operation maps each byte of the state to a new byte using S-box
    - the S-box is a 16×16 byte matrix computed using a formula
    - it was designed to resist known cryptanalytic attacks



ShiftRows

- The operation performs circular left shift on state rows
  - 2nd row is shifted by 1 byte
  - 3rd row is shifted by 2 bytes
  - 4th row is shifted by 3 bytes

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

→

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

- Important because other operations operate on a single cell

MixColumns

- The operation multiplies the state by a fixed matrix

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

  - was designed to ensure good mixing among the bytes of each column
  - small coefficients 01, 02, and 03 are for implementation efficiency

Other Operations

- Decryption:
  - inverse S-box is used in SubBytes
  - inverse shifts are performed in ShiftRows
  - inverse multiplication matrix is used in MixColumns
- Key expansion:
  - was designed to resist known attacks and be efficient
  - knowledge of a part of the key or round key doesn't enable calculation of other key bits
  - round-dependent values are used in key expansion

Summary of Rijndael design

- Simple design but resistant to known attacks
- Very efficient on a variety of platforms including 8-bit and 64-bit platforms
- Highly parallelizable
- Had the highest throughput in hardware among all AES candidates
- Well suited for restricted-space environments (very low RAM and ROM requirements)
- Optimized for encryption (decryption is slower)

AES Hardware Implementation

- It's been long known that hardware implementations of AES are extremely fast
  - the speed of encryption is compared with the speed of disk read
- Harware implementations were initially unaccessible to the average user
- Intel introduced new AES instruction set (AES-NI) in its commodity processors
  - other processor manufacturers support it now as well
  - hardware acceleration can be easily used on many platforms

Secure Encryption

Using a strong block cipher is not enough for secure encryption

- If you are to send more than 1 block (16 bytes) over the key lifetime, applying plain block cipher to the message as

$$\mathsf{Enc}_k(b_1), \mathsf{Enc}_k(b_2), \ldots$$

  will fail even weak definitions of secure encryption

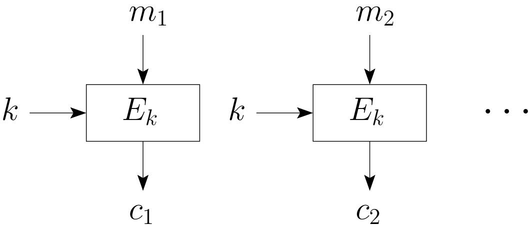
- No deterministic encryption can be secure if multiple blocks are sent

Encryption Modes

- Encryption modes indicate how messages longer than one block are encrypted and decrypted
- 4 modes of operation were standardized in 1980 for Digital Encryption Standard (DES)
  - can be used with any block cipher
  - electronic codebook mode (ECB), cipher feedback mode (CFB), cipher block chaining mode (CBC), and output feedback mode (OFB)
- 5 modes were specified with the current standard Advanced Encryption Standard (AES) in 2001
  - the 4 above and counter mode

Electronic Codebook (ECB) mode

- Divide the message m into blocks $m_1 m_2 \ldots m_\ell$ of size n each
- Encipher each block separately: for $i = 1, \ldots, \ell$, $c_i = E_k(m_i)$, where E denotes block cipher encryption
- The resulting ciphertext is $c = c_1 c_2 \ldots c_\ell$



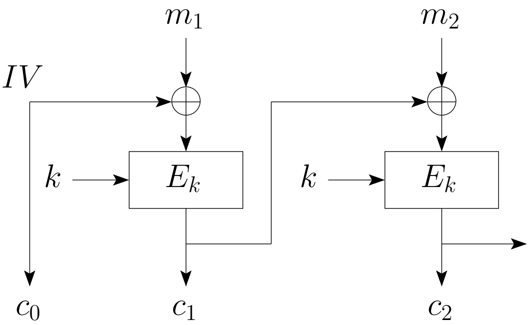
ECB Mode

ECB is a plain invocation of the block cipher

- Identical plaintext blocks result in identical ciphertexts (under the same key)
- This mode doesn't result in secure encryption
- It allows the block cipher to be used in other, more complex cryptographic constructions

Cipher Block Chaining (CBC) Mode

- Set $c_0 = IV \stackrel{R}{\leftarrow} \{0,1\}^n$ (initialization vector)
- Encryption: for $i = 1, \ldots, \ell$, $c_i = E_k(m_i \oplus c_{i-1})$
- Decryption: for $i = 1, \ldots, \ell$, $m_i = c_{i-1} \oplus D_k(c_i)$, where D is block cipher decryption



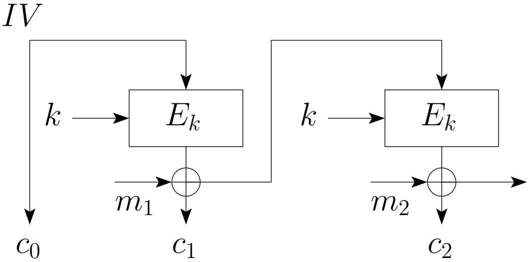
CBC Mode

Properties of the CBC mode:

- this mode is CPA-secure (has a formal proof) if the block cipher can be assumed to produce pseudorandom output
- a ciphertext block depends on all preceding plaintext blocks
- sequential encryption, cannot use parallel hardware
- IV must be random and communicated intact
  - if the IV is not random, security quickly degrades
  - if someone can fool the receiver into using a different IV, security issues arise

Cipher Feedback (CFB) Mode

- The message is XORed with the encryption of the feedback from the previous block
- Generate random IV and set initial input $I_1 = IV$
- Encryption: $c_i = E_k(I_i) \oplus m_i$; $I_{i+1} = c_i$
- Decryption: $m_i = c_i \oplus E_k(I_i)$

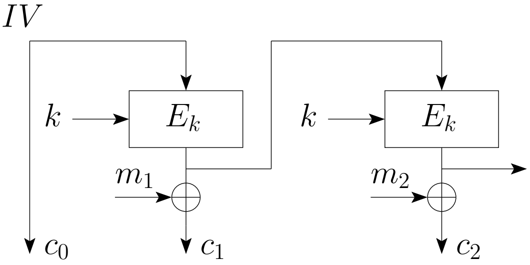


CFB Mode

Properties of the CFB mode:

- the mode is CPA-secure (under the same assumption that the block cipher is strong)
- similar to CBC, a ciphertext block depends on all previous plaintext blocks

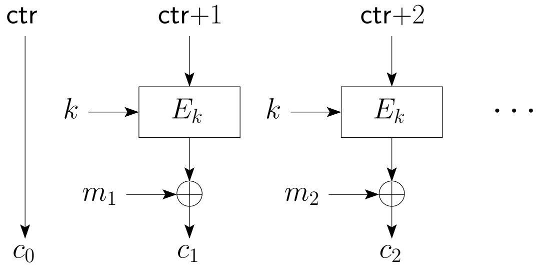
Output Feedback (OFB) Mode



- the feedback is independent of the message
- the mode is CPA-secure

Counter (CTR) Mode

- A counter is encrypted and XORed with a plaintext block
- No feedback into the encryption function
- Initially set $\mathsf{ctr} = IV \stackrel{R}{\leftarrow} \{0,1\}^n$



CTR Mode

Counter (CRT) mode

- encryption: for $i = 1, \ldots, \ell$, $c_i = E_k(\mathsf{ctr} + i) \oplus m_i$
- decryption: for $i = 1, \ldots, \ell$, $m_i = E_k(\mathsf{ctr} + i) \oplus c_i$

Properties:

- there is no need to pad the last block to full block size
- if the last plaintext block is incomplete, we just truncate the last cipherblock and transmit it

CTR Mode

Advantages of the counter mode

- Hardware and software efficiency: multiple blocks can be encrypted or decrypted in parallel
- Preprocessing: encryption can be done in advance; the rest is only XOR
- Random access: ith block of plaintext or ciphertext can be processed independently of others
- Security: at least as secure as other modes (i.e., CPA-secure)
- Simplicity: doesn't require decryption or decryption key scheduling

The counter can't be reused

Randomness Generation

- All cryptographic constructions that are non-deterministic or produce key material require randomness
  - key generation for any type of cryptographic tool
  - drawing randomness during probabilistic encryption
- What do we expect from a random bit sequence?
  - uniform distribution: all possible values are equally likely
  - independence: no part of the sequence depends on its other parts
- Where do we find randomness?

Randomness Generations

- Randomness can be gathered from physical, unpredictable processes
- Example sources of true randomness
  - least significant bits of time between key strokes
  - noise from a mouse, video camera, and microphone
  - variation in response times of raw read requests from a disk
- Amount of required randomness may not be small
  - example: choosing a 1024-bit prime
- Instead of a true randomness we can use a pseudo-random generator (PRG)

Pseudo-Random Generators

- A pseudo-random generator is an algorithm that
  - takes a short value, called a seed, as its input
  - produces a long string that is statistically close to a uniformly chosen random string
  - the bitlength of a seed is based on a security parameter
- The security requirement is that
  - a computationally bounded adversary cannot tell the output of a PRG apart from a truly random string of the same size
  - in practice, a number of statistical tests are used to test the strength of a PRG

Pseudo-Random Generators

PRGs are deterministic

- the output is always the same on the same seed
- for cryptographic purposes, it is crucial that the seed is hard to guess
  - i.e., use strong true randomness to generate a seed

Example of a PRG

- symmetric block ciphers, such as AES, can be used as PRGs
- given a key k (seed), produce a stream as $\mathsf{Enc}_k(0), \mathsf{Enc}_k(1), \ldots$, where Enc is block cipher encryption

Randomness in Implementations

- Regardless of how randomness was produced, it is absolutely crucial that you use good randomness
  - insufficient amount of randomness leads to predictable keys
  - this is especially dangerous for long-term signing keys
- Examples of poor randomness in cryptographic applications
  - CVE-2006-1833: Intel RNG Driver in NetBSD may always generate the same random number, Apr. 2006
  - CVE-2007-2453: Random number feature in Linux kernel does not properly seed pools when there is no entropy, Jun. 2007
  - CVE-2008-0166: OpenSSL on Debian-based operating systems uses a random number generator that generates predictable numbers, Jan. 2008

Putting It All Together

- AES is the current block cipher standard for symmetric encryption
  - it offers strong security and fast performance
- Secure encryption requires the use of a secure encryption mode
  - any mode except ECB is acceptable
  - the counter mode has attractive properties
- Strong randomness is required for cryptographic purposes
  - key generation, IV generation, etc.
- Implementing cryptographic constructions is hard
  - bugs exist even in well-known cryptographic libraries

Putting It All Together

When we store data encrypted, what do we do with the key?

Putting It All Together

When we store data encrypted, what do we do with the key?

What about integrity of the encrypted storage?