Advanced Cryptology

Padding

Ing. Josef Kokeš

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Příprava studijních programů Informatika pro novou fakultu ČVUT je spolufinancována Evropským sociálním fondem a rozpočtem Hlavního města Prahy v rámci Operačního programu Praha — adaptabilita (OPPA) projektem CZ.2.17/3.1.00/31952 – „Příprava a zavedení nových studijních programů Informatika na ČVUT v Praze“.
Praha & EU: Investujeme do vaší budoucnosti

Padding

Padding

- Padding is a cryptgraphic technique which lets us achieve some desirable results through the modifications of plaintext. The encryption algorithm itself is unchanged, but the pre- and postprocessing of the plaintext is different from the standard operation of the encryption/decryption algorithm.
- Typical goals:
  - Masking of specific properties of the plaintext (length, structure), prevention of usage of certain cryptanalytical methods.
  - Achieving a certain length of the plaintext, e.g. for the purpose of using specific block modes of operation or hash functions.
  - Protection against malleability of ciphers.
  - Assuring authenticity of messages (MAC = message authentication code)
- Beware! An incorrect use of padding can lead to a successful recovery of the plaintext!

Masking of the properties of the plaintext (1)

Masking of the properties of the plaintext

- Historical note: Messages often exhibit a structure which is known to the attacker or can be easily guessed. E.g. letters tend to start with a date, a salutation and the name of the recipient, first column of tabular data often is predictable (e.g. a date for a list of stock prices), naval messages often begin with an ID of the ship and the current time and position, etc. Padding can be used to move these almost-fixed parts of the message to a different place in the message. New symbols can be inserted into predetermined places of the message to confuse the attacker (e.g. to prevent a frequency analysis of the Caesar's cipher).
- These techniques are being used even today.

Masking of the properties of the plaintext (2)

Masking of the properties of the plaintext - examples

- Example: TOR (The Onion Router) - an anonymization network whose client generates and sends random messages to random nodes in order to mask an existence of the real message (the attacker can't determine whether a captured packet is a part of a communication or of the generated noise) and the recipient of a suspected communication (because messages are constantly being sent to different nodes, even if there is no real communication).
- Another example: VoIP protocols typically make use of compression. But silence is easier to compress than voice, and certain vowels are easier to compress than other vowels. An attacker could try to relate the changes in packet sizes into guesses about the content of the communication, or at least into assumptions about which party is talking at each particular moment. If we add extra noise into the communication, we can effectively counter these attempts.

Achieving a certain length of the plaintext (1)

Achieving a certain length of the plaintext

- Block ciphers and hash functions work with fixed-length blocks. But the message can be of any length, even such that it is not divisible by the block length.
- Solution: We pad the last block with extra data.
- Naive approaches:
  - Pad with random data: The padding becomes a part of the plaintext and we can't determine, what is the original plaintext and what is the padding (e.g., we decrypt `Price = 1234$a` - is `a` the padding and 1234 USD the price, or `$a` the padding and 1234 CZK the price? Or perhaps the padding is `4$a`?). Furthermore, consider that we are calculating a hash value of a file; with random padding, the hash will be different for each calculation, even thought the file didn't change.
  - All zero (or another fixed symbol) padding: Same as above: What if the plaintext itself contains zeroes? Also, with hash function we lose collision resistance - attacker can generate other messages with the same hash value by adding zeroes the the original message.

Achieving a certain length of the plaintext (2)

Requirements for a real solution

- We must be able to remove the padding
- The padding should be deterministic (for some uses, anyway)
- Padding should not provide the attacker with any information about the plaintext.

Achieving a certain length of the plaintext (3)

Bit padding

- Assume blocks n bits long, plaintext PT m bits long, $m \leq n$.
- Bit padding has the form 1000...00, with $n - m - 1$ zero bits.
- We append it to the end of the PT.
- Removal: We search the PT for the right-most 1. Everything from this point to the end of the PT is padding and can be removed.
- What if $n = m$? Then we would remove 1 or more bits of the message!
- Standard solution (used in most padding schemes): Add a new block which contains only padding.

Achieving a certain length of the plaintext (4)

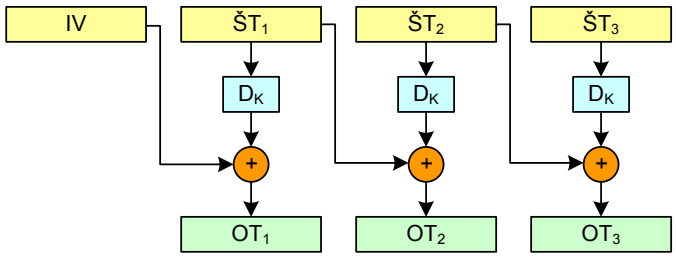
Byte padding

- Assume blocks n bytes long, plaintext PT m bytes long, $m < n$.
- A great number of schemes exist:
- ANSI X.923: Padding is formed as `00 00 ... 00 x`, where $x = n - m$ and the number of zero bytes is equal to $x - 1$. Removal requires that we read the last byte and remove that many bytes from the end of the message.
- ISO 10126: Similar to ANSI X.923, but uses random bytes in place of zero bytes. Note that this scheme does not satisfy the requirement for deterministic padding!
- PKCS7: Padding is formed as `x x ... x`, where $x = n - m$ and the length of padding is also x. I.e., padding could be `01`, `02 02`, `03 03 03` etc. Removal is the same as with ANSI X.923.
- ISO/IEC 7816-4: Application of the bit padding to bytes; the first byte of padding is `80`, the others `00`. Removal is the same as with the bit padding.
- Etc.

Padding Oracle Attack (1)

Padding Oracle Attack

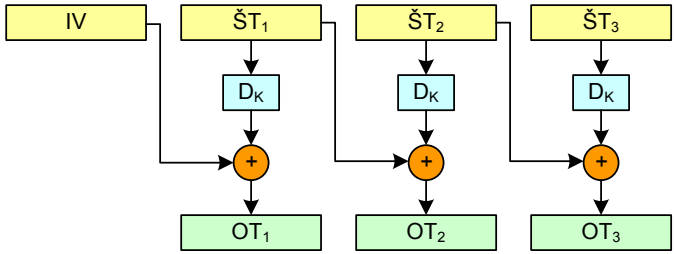
- Assume that we have a computer which is internally decrypting messages encrypted by some block cipher in the CBC mode. This computer first decrypts the messages and only then checks their integrity or content, and makes it possible to distinguish a failure in padding from a failure in integrity [1]. Then we can use this computer to decrypt messages sent by other people if we capture these messages, modify them and present them to the computer. We don't even need to know the key or even the cipher used!
- Recall: The CBC mode of operation:



- The attacker knows IV, ST_1, ST_2, ST_3 and the padding scheme (we will use PKCS7) and wants to learn OT_2.

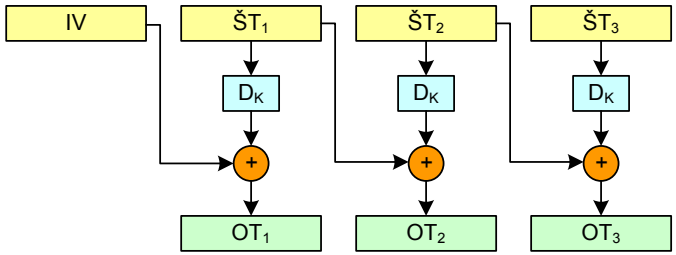
[1] E.g. by returning different error messages, or through timing differences - a failure in the padding exhibits in a shorter time than a failure in integrity of the actual message.

Padding Oracle Attack (2)



- The attacker knows IV, ST_1, ST_2, ST_3 (through packet capture) and the padding scheme (we will use PKCS7) and wants to learn OT_2.
- The attacker makes a guess that the last byte of OT_2 (we will call it a_1) has a value b_1. He changes the last byte of ST_1 from c_1 (which he knows) to $c_1 \oplus b_1 \oplus 01$, obtaining ST_A. When the victim attempts to decrypt ST_2, the last byte of OT_2 becomes $a_1 \oplus b_1 \oplus 01$. If the guess was correct and $a_1 = b_1$, this last byte will be `01` and will become a valid one-byte padding (but the message as a whole will be invalid). The attacker sends (IV, ST_1, ST_A) (discarding ST_3) to the padding oracle and checks for the result: If she receives a padding failure, then $a_1 \neq b_1$, she changes the value of b_1 and tries again. At most after 256 tries she receives an integrity or message failure, signifying that the padding of correct. At this moment she knowns the correct value of b_1 such that $a_1 = b_1$.

Padding Oracle Attack (3)



- The attacker now repeats the process for the second-to-last byte of OT_2 (a_2) and a two-byte padding $(02, 02)$ (that is, for all b_2 she changes c_2 to $c_2 \oplus b_2 \oplus 02$ and c_1 to $c_1 \oplus a_1 \oplus 02$ and sends (IV, ST_1, ST_A) to the oracle), etc. for all bytes of OT_2.
- Note that the complexity is only $256 \cdot m$ (where m is the block length in bytes), while brute-force attack at the cipher has complexity 256^m!
- Now she can repeat the process for OT_1, by modifying IV and sending (IV, ST_1) to the oracle.

Padding Oracle Attack (4)

Defense against Padding Oracle

- Modify the server so that the padding failures are indistinguishable from message/integrity failures. Note that this can be quite difficult with timing attacks, because an optimizing compiler will remove „unnecessary“ loops in order to achieve better performance.
- Modify the server so that it first verifies the integrity of the whole encrypted message (including encrypted padding) and then attempt decryption and padding removal. (i.e. se the „encrypt-then-MAC“ method rather than „MAC-then-encrypt“). Note: Removing the integrity checks won’t help if the attacker can recognize whether the padding was or was not correct.
- Use a cipher which doesn’t use padding (e.g. a stream cipher).
- Use an encryption mode which doesn’t use padding (e.g. CTR, the counter mode).

Padding and asymmetric ciphers (1)

Padding and asymmetric ciphers

- Asymmetric cipher is, for the purpose of padding, very similar to regular symmetric block ciphers, although we speak of the module length, not block length.
- Asymmetric ciphers are malleable: an attacker can introduce predictable changes to the plaintext without breaking the cipher.
- Specifically for RSA, some ciphertexts may be easy to decrypt if we don't use padding.
- If we don't use padding, RSA as it is often used is a deterministic transformation, i.e. the same plaintext always leads to the same ciphertext.

Padding and asymmetric ciphers (2)

Malleability of El Gamal

- Recall: Let x be the private key, $(m, g, c = g^x)$ the public key, p the plaintext and y nonce chosen by the sender. Then $CT = (d = |g^y|_m, e = |c^y \cdot p|_m)$ is the ciphertext of El Gamal. We can decrypt it using $c^y = |(g^x)^y|_m = |(g^y)^x|_m = |d^x|_m, C = |(c^y)^{-1}|_m, p' = |C \cdot e|_m = |(c^y)^{-1} \cdot c^y \cdot p|_m = p$.
- The attacker can modify the CT to inject a predictable change to p:
- $CT' = (d, s \cdot e) \Rightarrow p' = s \cdot p$
- Notice that the attacker doesn't need to know the value of p, and still can multiply p by a constant. If p represents some text, the recipient will likely notice the change, but if p represents an integer number (e.g. a price, account number, phone number...), this change might pass unnoticed.

Padding and asymmetric ciphers (3)

Malleability of RSA

- Recall: Let d be the private key, (n, e) the public key such that $|d \cdot e|_{\Phi(n)} = 1$, p the plaintext. Then $CT = |p^e|_n$ is the ciphertext of RSA. We can decrypt it using $|CT^d|_n = p$.
- The attacker can modify the CT to inject a predictable change to p:
- $CT' = |s^e \cdot CT \Rightarrow p'|n = |CT'^d|n = |(s^e \cdot p^e)^d|n = s \cdot p$
- Once again, the attacker modified the plaintext p in a predictable manner (multiplied it by a constant s) without being able to decrypt it first.

Padding and asymmetric ciphers (4)

Decrypting the RSA ciphertext

- Assume a public key ($2^{2047} < n < 2^{2048}, e = 3$)
- Let $p \approx 2^{512}$.
- Then $CT = |p^e|_n \approx |2^{1536}|_n \approx 2^{1536}$.
- The attacker knows $e = 3$ (public key). She tries to calculate $\sqrt[3]{ST}$ in real numbers, where it is very simple. Indeed, she receives $\sqrt[3]{p^3} = p$. If we have a large key, small exponent and sufficiently small plaintext, we can decrypt the plaintext very quickly (here we got 512 bits = 64 bytes of the plaintext)!

Padding and asymmetric ciphers (5)

Deterministic transformation of RSA

- The RSA encryption follows the formula $CT = |p^e|_n$. n is constant, e is constant, it follows that $p_1 = p_2 \Leftrightarrow CT_1 = CT_2$
- The attacker can recognize the repeated blocks in exactly the same way as she did when the victims used ECB encryption mode. The statistical properties of the repeated blocks can be used to deduce some information about the plaintext.

Padding and asymmetric ciphers (6)

Padding to the rescue: Malleability

- Padding becomes a part of the plaintext p, i.e. $p = PT \odot PD$, where PT is the actual plaintext, PD is the padding and $\odot$ represents string concatenation.
- This results in padding becoming a part of the ciphertext:
- $CT_{ElGamal} = (d = |g^y|_m, e = |c^y \cdot p|_m) = (d = |g^y|_m, e = |c^y \cdot (PT \odot PD)|_m)$
- $CT_{RSA} = |p^e|_m = |(PT \odot PD)^e|_m$
- If the attacker attempts to change the value of p, she inevitably changes both PT and PD. Since PD is deterministic, the victim can detect that it got a modified ciphertext and should discard the resulting plaintext.

Padding and asymmetric ciphers (7)

Padding to the rescue: Fast decryption of RSA

- Non-zero padding increases the bitlength of p by up to the length of PD (if we use a padding in the form, e.g., 1000...0001). We can design the length of PD to that even if we exponentiate the modified p by a small exponent e, we will overflow the n bits of the modulus and force the modulo operation, which will stop the fast real root from decrypting the message.

Padding and asymmetric ciphers (8)

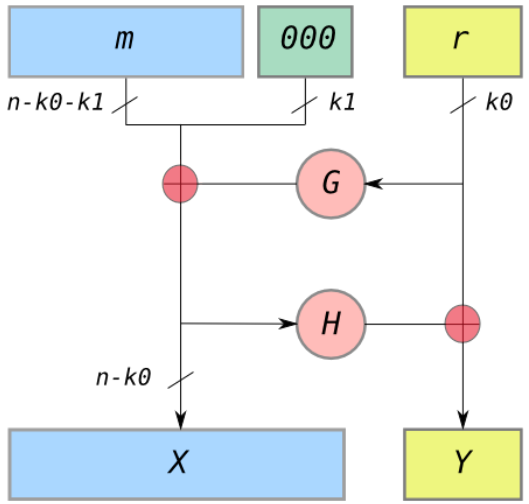
Padding to the rescue: Deterministic transformation of RSA

- We can choose such a padding scheme where a part of the information is randomly generated. Then each encryption will use a different value of PD, which ensures a different value of $p = PT \odot PD$, even if we keep encrypting the same PT. Then also the $CT = |p^e|_m$ will have a different value.
- We need to consider the tradeoff between the deterministic and the non-deterministic part of padding: The deterministic part prevents malleability, the non-deterministic part solves the problem of deterministic transformation.
- A nice implementation of this tradeoff is the Optimal Asymmetric Encryption Padding scheme.

Padding and asymmetric ciphers (9)

Optimal Asymmetric Encryption Padding

- Assume an asymmetric[2] block cipher with a block of length n bits.
- Select constants k_0, k_1 such that $k_0 + k_1 < n$, and random oracles $G : \{0,1\}^{k_0} \longrightarrow \{0,1\}^{n-k_0}$ and $H : \{0,1\}^{n-k_0} \longrightarrow \{0,1\}^{k_0}$. These parameters define the following padding scheme:



3

[2] Why asymmetric?

[3] Source: OAEP diagram, Wikipedia Commons, http://en.wikipedia.org/wiki/File:Oaep-diagram-20080305.png

Padding and asymmetric ciphers (10)

Optimal Asymmetric Encryption Padding

- Divide the plaintext into blocks m_i of length $n - k_0 - k_1$ (not n as was the case with regular asymmetric cipher). r_i is a randomly-generated content. Calculate X_i and Y_i, their concatenation $X_i \odot Y_i$ becomes the input (plaintext) of the asymmetric cipher.
- Note: OAEP has a Feistel-network structure.
- Properties:
  - The random part solves the problem of deterministic transformation.
  - The scheme prevents partial decryption (to extract m from X and Y, we need to decrypt the whole X and Y). We can verify the correct removal of padding by checking that the plaintext contains k_1 zero bits at the right position.
  - Instead of the k_1 zero bits, we can also use a suitable MAC (e.g. HMAC). Of course, this MAC will run in the MAC-then-encrypt order $\Rightarrow$ beware of the Padding Oracle!!

Padding and message integrity

Padding and message integrity

- Some padding schemes can incorporate MAC, e.g. modified OAEP.
- Some padding schemes can let an attacker generate fake messages with a valid MAC. This is generally true for schemes such as zero-padding where we can't properly determine where the padding begins.
- Assume a MAC-generating function H with input message n bits long. Let p be a message of length $m < n$ bits.
- Assume that we use zero-padding, i.e. we add $n - m$ zero bits to the message: $p' = p \odot 000...000$. Then certainly $H(p) = H(p \odot 0) = H(p \odot 00) = ...$
- The attacker can generate up to $n - m$ new messages and append the MAC calculated for the original message. It will be valid for all fake messages, too.