Padding (cryptography)

In cryptography, **padding** is any of a number of distinct practices which all include adding data to the beginning, middle, or end of a message prior to encryption. In classical cryptography, padding may include adding nonsense phrases to a message to obscure the fact that many messages end in predictable ways, e.g. *sincerely yours*.

Contents

Classical cryptography

Official messages often start and end in predictable ways: *My dear ambassador, Weather report, Sincerely yours*, etc. The primary use of padding with classical ciphers is to prevent the cryptanalyst from using that predictability to find known plaintext[1] that aids in breaking the encryption. Random length padding also prevents an attacker from knowing the exact length of the plaintext message.

A famous example of classical padding which caused a great misunderstanding is "the world wonders" incident, which nearly caused an Allied loss at the WWII Battle off Samar, part of the larger Battle of Leyte Gulf. In that example, Admiral Chester Nimitz, the Commander in Chief, U.S. Pacific Fleet in World War II, sent the following message to Admiral Bull Halsey, commander of Task Force Thirty Four (the main Allied fleet) at the Battle of Leyte Gulf, on October 25, 1944:[2]

> Where is, repeat, where is Task Force Thirty Four?[3]

With padding (bolded) and metadata added, the message became:

> **TURKEY TROTS TO WATER** GG FROM CINCPAC ACTION COM THIRD FLEET INFO COMINCH CTF SEVENTY-SEVEN X WHERE IS RPT WHERE IS TASK FORCE THIRTY FOUR RR **THE WORLD WONDERS**[3]

Halsey's radio operator mistook some of the padding for the message, so Admiral Halsey ended up reading the following message:

> Where is, repeat, where is Task Force Thirty Four? The world wonders[3]

Admiral Halsey interpreted the padding phrase "the world wonders" as a sarcastic reprimand, causing him to have an emotional outburst and then lock himself in his bridge and sulk for an hour before moving his forces to assist at the Battle off Samar.[2] Halsey's radio operator should have been tipped off by the letters *RR* that "the world wonders" was padding; all other radio operators who received Admiral Nimitz's message correctly removed both padding phrases.[2]

Many classical ciphers arrange the plaintext into particular patterns (e.g., squares, rectangles, etc.) and if the plaintext doesn't exactly fit, it is often necessary to supply additional letters to fill out the pattern. Using nonsense letters for this purpose has a side benefit of making some kinds of cryptanalysis more difficult.

Symmetric cryptography

Hash functions

Most modern cryptographic hash functions process messages in fixed-length blocks; all but the earliest hash functions include some sort of padding scheme. It is critical for cryptographic hash functions to employ termination schemes that prevent a hash from being vulnerable to length extension attacks.

Many padding schemes are based on appending predictable data to the final block. For example, the pad could be derived from the total length of the message. This kind of padding scheme is commonly applied to hash algorithms that use the Merkle–Damgård construction such as MD-5, SHA-1, and SHA-2 family such as SHA-224, SHA-256, SHA-384, SHA-512, SHA512/224, and SHA-512/256[4]

Block cipher mode of operation

Cipher-block chaining (CBC) mode is an example of block cipher mode of operation. Some block cipher modes (CBC and PCBC essentially) for symmetric-key encryption algorithms require plain text input that is a multiple of the block size, so messages may have to be padded to bring them to this length.

There is currently a shift to use streaming mode of operation instead of block mode of operation. An example of streaming mode encryption is the counter mode of operation.[5] Streaming modes of operation can encrypt and decrypt messages of any size and therefore do not require padding. More intricate ways of ending a message such as ciphertext stealing or residual block termination avoid the need for padding.

A disadvantage of padding is that it makes the plain text of the message susceptible to padding oracle attacks. Padding oracle attacks allow the attacker to gain knowledge of the plain text without attacking the block cipher primitive itself. Padding oracle attacks can be avoided by making sure that an attacker cannot gain knowledge about the removal of the padding bytes. This can be accomplished by verifying a message authentication code (MAC) or digital signature *before* removal of the padding bytes, or by switching to a streaming mode of operation.

Bit padding

Bit padding can be applied to messages of any size.

A single set ('1') bit is added to the message and then as many reset ('0') bits as required (possibly none) are added. The number of reset ('0') bits added will depend on the block boundary to which the message needs to be extended. In bit terms this is "1000 ... 0000".

This method can be used to pad messages which are any number of bits long, not necessarily a whole number of bytes long. For example, a message of 23 bits that is padded with 9 bits in order to fill a 32-bit block:

```
... | 1011 1001 1101 0100 0010 0111 0000 0000 |
```

This padding is the first step of a two-step padding scheme used in many hash functions including MD5 and SHA. In this context, it is specified by RFC1321 (http://www.faqs.org/rfcs/rfc1321.html) step 3.1.

This padding scheme is defined by ISO/IEC 9797-1 as Padding Method 2.

Byte padding

Byte padding can be applied to messages that can be encoded as an integral number of bytes.

ANSI X9.23

In ANSI X9.23, between 1 and 8 bytes are always added as padding. The block is padded with random bytes (although many implementations use 00) and the last byte of the block is set to the number of bytes added.[6]

Example: In the following example the block size is 8 bytes, and padding is required for 4 bytes (in hexadecimal format)

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 00 00 00 04 |
```

ISO 10126

ISO 10126 (withdrawn, 2007[7][8]) specifies that the padding should be done at the end of that last block with random bytes, and the padding boundary should be specified by the last byte.

Example: In the following example the block size is 8 bytes and padding is required for 4 bytes

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 81 A6 23 04 |
```

PKCS#5 and PKCS#7

PKCS#7 is described in RFC 5652 (https://tools.ietf.org/html/rfc5652#section-6.3).

Padding is in whole bytes. The value of each added byte is the number of bytes that are added, i.e. *N* bytes, each of value *N* are added. The number of bytes added will depend on the block boundary to which the message needs to be extended.

The padding will be one of:

```
01
02 02
03 03 03
04 04 04 04
05 05 05 05 05
06 06 06 06 06 06
etc.
```

This padding method (as well as the previous two) is well-defined if and only if *N* is less than 256.

Example: In the following example the block size is 8 bytes and padding is required for 4 bytes

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 04 04 04 04 |
```

If the length of the original data is an integer multiple of the block size *B*, then an extra block of bytes with value *B* is added. This is necessary so the deciphering algorithm can determine with certainty whether the last byte of the last block is a pad byte indicating the number of padding bytes added or part of the plaintext message. Consider a plaintext message that is an integer multiple of *B* bytes with the last byte of plaintext being **01**. With no additional information, the deciphering algorithm will not be able to determine whether the last byte is a plaintext byte or a pad byte. However, by adding *B* bytes each of value *B* after the **01** plaintext byte, the deciphering algorithm can always treat the last byte as a pad byte and strip the appropriate number of pad bytes off the end of the ciphertext; said number of bytes to be stripped based on the value of the last byte.

PKCS#5 padding is identical to PKCS#7 padding, except that it has only been defined for block ciphers that use a 64-bit (8-byte) block size. In practice the two can be used interchangeably.

ISO/IEC 7816-4

ISO/IEC 7816-4:2005[9] is identical to the bit padding scheme, applied to a plain text of *N* bytes. This means in practice that the first byte is a mandatory byte valued '80' (Hexadecimal) followed, if needed, by 0 to $N - 1$ bytes set to '00', until the end of the block is reached. ISO/IEC 7816-4 itself is a communication standard for smart cards containing a file system, and in itself does not contain any cryptographic specifications.

Example: In the following example the block size is 8 bytes and padding is required for 4 bytes

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 80 00 00 00 |
```

The next example shows a padding of just one byte

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD DD DD DD 80 |
```

Zero padding

All the bytes that are required to be padded are padded with zero. The zero padding scheme has not been standardized for encryption, although it is specified for hashes and MACs as Padding Method 1 in ISO/IEC 10118-1[10] and ISO/IEC 9797-1.[11]

Example: In the following example the block size is 8 bytes and padding is required for 4 bytes

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 00 00 00 00 |
```

Zero padding may not be reversible if the original file ends with one or more zero bytes, making it impossible to distinguish between plaintext data bytes and padding bytes. It may be used when the length of the message can be derived out-of-band. It is often applied to binary encoded strings (null-terminated string) as the null character can usually be stripped off as whitespace.

Zero padding is sometimes also referred to as "null padding" or "zero byte padding". Some implementations may add an additional block of zero bytes if the plaintext is already divisible by the block size.

Public key cryptography

In public key cryptography, padding is the process of preparing a message for encryption or signing using a specification or scheme such as PKCS#1 v1.5, OAEP, PSS, PSSR, IEEE P1363 EMSA2 and EMSA5. A modern form of padding for asymmetric primitives is OAEP applied to the RSA algorithm, when it is used to encrypt a limited number of bytes.

The operation is referred to as "padding" because originally, random material was simply appended to the message to make it long enough for the primitive. This form of padding is not secure and is therefore no longer applied. A modern padding scheme aims to ensure that the attacker cannot manipulate the plaintext to exploit the mathematical structure of the primitive and will usually be accompanied by a proof, often in the random oracle model, that breaking the padding scheme is as hard as solving the hard problem underlying the primitive.

Traffic analysis and protection via padding

Even if perfect cryptographic routines are used, the attacker can gain knowledge of the amount of traffic that was generated. The attacker might not know what Alice and Bob were talking about, but can know that they *were* talking and *how much* they talked. In some circumstances this leakage can be highly compromising. Consider for example when a military is organising a secret attack against another nation: it may suffice to alert the other nation for them to know merely that there *is* a lot of secret activity going on.

As another example, when encrypting Voice Over IP streams that use variable bit rate encoding, the number of bits per unit of time is not obscured, and this can be exploited to guess spoken phrases.[12] Similarly, the burst patterns that common video encoders produce are often sufficient to identify the streaming video a user is watching uniquely.[13] Even the *total size* of an object alone, such as a website, file, software package download, or online video, can uniquely identify an object, if the attacker knows or can guess a known set the object comes from.[14][15][16] The side-channel of encrypted content length was used to extract passwords from HTTPS communications in the well-known CRIME and BREACH attacks.[17]

Padding an encrypted message can make traffic analysis harder by obscuring the true length of its payload. The choice of length to pad a message to may be made either deterministically or randomly; each approach has strengths and weaknesses that apply in different contexts.

Randomized padding

A random number of additional padding bits or bytes may be appended to the end of a message, together with an indication at the end how much padding was added. If the amount of padding is chosen as a uniform random number between 0 and some maximum M, for example, then an eavesdropper will be unable to determine the message's length precisely within that range. If the maximum padding M is small compared to the message's total size, then this padding will not add much overhead, but the padding will obscure only the least-significant bits of the object's total length, leaving the approximate length of large objects readily observable and hence still potentially uniquely identifiable by their length. If the maximum padding M is comparable to the size of the payload, in contrast, an eavesdropper's uncertainty about the message's true payload size is much larger, at the cost that padding may add up to 100% overhead (2× blow-up) to the message.

In addition, in common scenarios in which an eavesdropper has the opportunity to see *many* successive messages from the same sender, and those messages are similar in ways the attacker knows or can guess, then the eavesdropper can use statistical techniques to decrease and eventually even eliminate the benefit of randomized padding. For example, suppose a user's application regularly sends messages of the same length, and the eavesdropper knows or can guess fact based on fingerprinting the user's application for example. Alternatively, an active attacker might be able to *induce* an endpoint to send messages regularly, such as if the victim is a public server. In such cases, the eavesdropper can simply compute the average over many observations to determine the length of the regular message's payload.

Deterministic padding

A deterministic padding scheme always pads a message payload of a given length to form an encrypted message of a particular corresponding output length. When many payload lengths map to the same padded output length, an eavesdropper cannot distinguish or learn any information about the payload's true length within one of these length *buckets*, even after many observations of the identical-length messages being transmitted. In this respect, deterministic padding schemes have the advantage of not leaking any additional information with each successive message of the same payload size.

On the other hand, suppose an eavesdropper can benefit from learning about *small* variations in payload size, such as plus or minus just one byte in a password-guessing attack for example. If the message sender is unlucky enough to send many messages whose payload lengths vary by only one byte, and that length is exactly on the border between two of the deterministic padding classes, then these plus-or-minus one

payload lengths will consistently yield different padded lengths as well (plus-or-minus one block for example), leaking exactly the fine-grained information the attacker desires. Against such risks, randomized padding can offer more protection by independently obscuring the least-significant bits of message lengths.

Common deterministic padding methods include padding to a constant block size and padding to the next-larger power of two. Like randomized padding with a small maximum amount M, however, padding deterministically to a block size much smaller than the message payload obscures only the least-significant bits of the messages true length, leaving the messages's true approximate length largely unprotected. Padding messages to a power of two (or any other fixed base) reduces the maximum amount of information that the message can leak via its length from $O(\log M)$ to $O(\log \log M)$. Padding to a power of two increases message size overhead by up to 100%, however, and padding to powers of larger integer bases increase maximum overhead further.

The PADMÉ scheme, proposed for padded uniform random blobs or PURBs, deterministically pads messages to lengths representable as a floating point number whose mantissa is no longer (i.e., contains no more significant bits) than its exponent.[16] This length constraint ensures that a message leaks at most $O(\log \log M)$ bits of information via its length, like padding to a power of two, but incurs much less overhead of at most 12% for tiny messages and decreasing gradually with message size.

See also

- Chaffing and winnowing, mixing in large amounts of nonsense before sending
- Ciphertext stealing, another approach to deal with messages that are not a multiple of the block length
- Initialization vector, salt (cryptography), which are sometimes confused with padding
- Key encapsulation, an alternative to padding for public key systems used to exchange symmetric keys
- PURB or *padded uniform random blob*, an encryption discipline that minimizes leakage from either metadata or length
- Russian copulation, another technique to prevent cribs

References

1. Gordon Welchman, *The Hut Six Story: Breaking the Enigma Codes*, p. 78.
2. Willmott, H. P. (19 August 2005). "The Great Day of Wrath: 25 October 1944". *The Battle of Leyte Gulf: The Last Fleet Action*. Indiana University Press. ISBN 9780253003515.
3. Tuohy, William (2007). *America's Fighting Admirals: Winning the War at Sea in World War II* (https://archive.org/details/americasfighting00tuoh). MBI Publishing Company. ISBN 9780760329856.
4. NIST. "FIPS 180-4 Secure Hash Standard (SHS)" (https://csrc.nist.gov/csrc/media/publications/fips/180/4/final/documents/fips180-4-draft-aug2014.pdf) (PDF). NIST..
5. https://www.cs.columbia.edu/~smb/classes/s09/l05.pdf, pg 17
6. "ANSI X9.23 cipher block chaining" (https://www.ibm.com/support/knowledgecenter/en/linuxonibm/com.ibm.linux.z.wskc.doc/wskc_c_l0wskc58.html). *IBM Knowledge Center*. IBM. Retrieved 31 December 2018.
7. ISO catalog, *ISO 10126-1:1991* (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=18113)
8. ISO catalog, *ISO 10126-2:1991* (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=18114)

9. ISO catalog, *ISO/IEC 7816-4:2005* (http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=36134)
10. ISO/IEC 10118-1:2000 *Information technology – Security techniques – Hash-functions – Part 1: General* (http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=31143)
11. ISO/IEC 9797-1:1999 *Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher* (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=30656)
12. Wright, Charles V.; Ballard, Lucas; Coull, Scott E.; Monrose, Fabian; Masson, Gerald M. (1 December 2010). "Uncovering Spoken Phrases in Encrypted Voice over IP Conversations". *ACM Transactions on Information and System Security*. **13** (4): 35. CiteSeerX 10.1.1.363.1973 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.363.1973). doi:10.1145/1880022.1880029 (https://doi.org/10.1145%2F1880022.1880029). S2CID 9622722 (https://api.semanticscholar.org/CorpusID:9622722).
13. Schuster, Roei; Shmatikov, Vitaly; Tromer, Eran (August 2017). *Beauty and the Burst: Remote Identification of Encrypted Video Streams* (https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/schuster). USENIX Security Symposium (https://www.usenix.org/conference/usenixsecurity17).
14. Hintz, Andrew (April 2002). *Fingerprinting Websites Using Traffic Analysis*. International Workshop on Privacy Enhancing Technologies. doi:10.1007/3-540-36467-6_13 (https://doi.org/10.1007%2F3-540-36467-6_13).
15. Sun, Qixiang; Simon, D.R.; Wang, Yi-Min; Russell, W.; Padmanabhan, V.N.; Qiu, Lili (May 2002). *Statistical Identification of Encrypted Web Browsing Traffic*. IEEE Symposium on Security and Privacy. doi:10.1109/SECPRI.2002.1004359 (https://doi.org/10.1109%2FSECPRI.2002.1004359).
16. Nikitin, Kirill; Barman, Ludovic; Lueks, Wouter; Underwood, Matthew; Hubaux, Jean-Pierre; Ford, Bryan (2019). "Reducing Metadata Leakage from Encrypted Files and Communication with PURBs" (https://petsymposium.org/2019/files/papers/issue4/popets-2019-0056.pdf) (PDF). *Proceedings on Privacy Enhancing Technologies (PoPETS)*. **2019** (4): 6–33. doi:10.2478/popets-2019-0056 (https://doi.org/10.2478%2Fpopets-2019-0056). S2CID 47011059 (https://api.semanticscholar.org/CorpusID:47011059).
17. Sheffer, Y.; Holz, R.; Saint-Andre, P. (February 2015). Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS) (https://tools.ietf.org/html/rfc7457) (Report).

Further reading

- XCBC: csrc.nist.gov/groups/ST/toolkit/BCM/documents/workshop2/presentations/xcbc.pdf (http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/workshop2/presentations/xcbc.pdf)