DUAL COUNTER MODE

MIKE BOYLE

CHRIS SALTER

INTRODUCTION

For the past 18 months, the NSA has been developing a high-speed encryption mode for IP packets. The mode that we designed is identical in many aspects to Jutla's Integrity Aware Parallelizable Mode (IAPM). There is one important difference in our proposal. In the IP world, a large number of packets might arrive out of order. Integrity Aware Parallelizable Mode (IAPM) and the proposed variations incur a large overhead for out of order packets[JU 01]. Each packet requires at least the time to perform a full decryption to obtain an IV before decryption of the cipher can begin. This note describes our solution to this problem.

First, we describe the basic mode and its features. We then describe how to implement this mode for IPSec.

DUAL COUNTER MODE

Dual counter mode is a hybrid of ECB mode and counter mode. Let E represent encryption by a codebook of width W. Let P_1, P_2, ..., P_j be j blocks of plaintext and let C_1, C_2, ..., C_j be the corresponding ciphertext. Let f be a polynomial of degree W for a primitive linear feedback shift register. Also, let $\{x_i\}$ be the sequence of fills generated by this polynomial. The first fill, x_0, is a secret shared between the two peers. This initial fill is most easily derived from the key exchange[1]. Dual counter mode can be described as follows:

j = # of datablocks

For i = 1, ..., j

$$x_i = f(x_{i-1})$$

$$C_i = E(P_i \oplus x_i) \oplus x_i$$

Quite likely the cipherblocks will travel in packets. If the packets arrive in order, the receiver does not lose track of the fill needed to decrypt the cipher.

TWO IMPLEMENTATION MODES

We knew that many implementers would want to verify the data integrity of packets. This mode has the property that any change to a ciphertext block causes the decrypted plaintext to be garbled. Thus it is easy to add a checksum to verify data integrity.

[1] Of course, care should be taken in producing this value. For example, the designers of the key exchange for IPsec used secure hashes such as SHA-1 to isolate keying material.

We developed this mode for high data rate IP encryption. In the second variation we suggest a possible implementation for IPsec that is efficient for packets.

VERIFYING DATA INTEGRITY

To detect attempts to make malicious changes to cipher, a simple checksum of the data can be encrypted and appended as an additional cipherblock.

checksum = 0

j = # of datablocks

For i = 1, ..., j

$x_i = f(x_{i-1})$

$C_i = E(P_i \oplus x_i) \oplus x_i$

checksum = checksum $\oplus$ P_i

$x_{j+1} = f(x_j)$

$C_{j+1} = E(\text{checksum} \oplus x_{j+1}) \oplus x_0$

Note that checking data integrity only costs one pipelined step of the codebook.

TACKLING OUT OF ORDER IP PACKETS

In the IP world, packets can arrive out of order. Encrypting the current fill and placing it in the packet's header would allow the decrypting device to obtain the correct fill for an out of order packet. This solution, however, defeats the whole point of doing a variation of counter mode, which is the ability to pipeline the codebook for the entire message.

Each IP packet has a 32-bit sequence number (SEQ). An IPsec packet also has a 32-bit security parameter index (SPI). The SPI and sequence number can be concatenated to produce a 64-bit block. Some padding might be required to create a value from the sequence number and SPI that is the same size as the width of the codebook. To produce a 128-bit padded sequence number for the AES algorithm, concatenate this 64-bit number with its complement. This number is added to the shared secret (x_0) and used as the initial register fill for that packet.

Thus the sender and receiver both set the current fill of the register at the beginning of the packet to be: x_0 + (SEQ|SPI|padding), where x_0 is the shared secret negotiated during the key exchange. To minimize the hardware and overhead, the addition could be done as a vector of 32-bit adds[2]. The

[2] Addition mod 2^{128} would be expensive to implement in hardware. Since common codebook widths are divisible by 32, a reasonable alternative is to break the numbers into 32-bit fields and to perform mod 2^{32} addition within the corresponding fields.

register steps forward from this per packet starting point to produce the sequence of fills needed for the data blocks[3].

The padded sequence number, which includes the SPI, also needs to be authenticated. This number and the plaintext blocks are exclusive-or added to produce a checksum the same size as the width of the codebook.

checksum = SEQ|SPI|padding

j = # of datablocks

$y_0 = x_0 + (SEQ|SPI|padding)$

For i = 1, …, j

$y_i = f(y_{i-1})$

$C_i = E(P_i \oplus y_i) \oplus y_i$

checksum = checksum $\oplus$ P_i

$y_{j+1} = f(y_j)$

$C_{j+1} = E(checksum \oplus y_{j+1}) \oplus y_0$

ANALYSIS

Adding the fill to the plaintext eliminates the undesirable property of ECB mode that repeated cipher results in repeated plaintext. Incorporating codebook mode by running the plaintext and a fill through the codebook eliminates bit-flipping attacks.

Adding the fill the second time to the subcipher helps prevent the adversary from doctoring the components of the linear checksum and thus also helps prevent cut-and-paste and replay attacks.

A shift register was chosen instead of a 1-up counter to prevent an adversary from reordering cipher blocks within a packet. For instance, adjacent blocks would have counters that differ only in the low order bit 1/2 the time. The adversary could add out that difference in the cipher when blocks are reordered, thus eliminating the difference sent into the codebook. The linear checksum would not pick up that the two blocks of plaintext were added in a different order. Using a shift register with hidden state hinders an adversary's ability to guess the difference between two fills.

A possible drawback of dual counter mode is that the entire packet must be decrypted before attempts to change the data are detected. Quite likely, however, decrypting the packet with a

[3] Although any primitive degree 128 polynomial would do, we suggest using $x^{128} + x^7 + x^2 + x + 1$ in a Galois configuration. Since there are no primitive trinomials of degree 128 this should be optimal in hardware, and having all the feedback taps in one byte should optimize it for software as well.

pipelined codebook should take no longer than applying an HMAC. Moreover, both authentication and encryption are completed at the same time.

COMPARISON WITH OTHER MODES

It is worth considering the implementation trade-offs between dual counter mode and the two other modes traditionally used in a high data rate environment, namely counter and CBC modes. Counter mode and dual counter mode allow pipelining, while CBC can only be pipelined in the decrypt direction. There is a technique for speeding up CBC mode that requires the use of additional initialization vectors. For example, to be twice as fast, two copies of the codebook and two initialization vectors are required. The odd number blocks are encrypted with the chain started by one initialization vector and the even by the other chain. We have not explored the nuances of how to implement this technique, such as whether each chain should be protected by a different key, since we have presented an alternative mode that does not have the data integrity problems of CBC.

Counter mode does not require the decrypt version of the codebook, whereas CBC and dual counter mode do. This may not be significant for three reasons. The first is that the decrypt implementation may be able to share circuitry with the encrypt version. The second reason is more straightforward: in trying to maximize throughput, there will probably be two copies of the codebook hardware, one for sending and one for receiving messages. Finally, space on the chip is freed up since an optimized hash implementation is not necessary for data integrity.

As mentioned in the introduction, it would be useful to have a version of Integrity Aware Parallelizable Mode (IAPM) that does not incur a large overhead for out of order packets[JU 01].

CONCLUSION

Several modes have been proposed to speed up IP encryption. Most eliminate the need to have a separate mode for data integrity. Our variation differs from these submissions by preserving the ability to parallelize even if packets arrive out of order.

REFERENCES

[Bel 96] Steven M. Bellovin. Problem Areas for the IP Security Protocols. In Proceedings of the Sixth Usenix UNIX Security Symposium, July 1996. Available at ftp://ftp.research.att.com/dist/smb/badesp.ps

[HC 98] D. Harkins and D. Carrel. The Internet Key Exchange (IKE), November 1998. RFC 2409.

[JU 01] C. Jutla. Encryption Modes with Almost Free Message Integrity. To appear in EUROCRYPT'2001.

[KA 98] S. Kent and R. Atkinson. IP Encapsulating Security Payload (ESP), November 1998. RFC 2406.

[MD 98] C. Madson and N. Doraswamy. The ESP DES-CBC Cipher Algorithm with Explicit IV, November 1998. RFC 2405.

[MOV 96] A. Menezes, P. Oorschot, S. Vanstone. Handbook of Applied Cryptography. CRC Press. 1996.

[Sch 96] Bruce Schneier. Applied Cryptography, 2nd edition. John Wiley and Sons. 1996

SUMMARY OF PROPERTIES

Security Function: dual encryption/data integrity mode for a codebook.

Error Propagation: an error in a cipher block causes all the data in the packet to fail the data integrity check. Cipher blocks without errors still decrypt correctly.

Synchronization: key exchange initializes mode and in a less reliable environment packets carry information necessary for decryption to proceed.

Parallelizability: mode allows encryption and decryption to be pipelined.

Keying Material Requirements: the mode requires that the key exchange produce an additional secret initialization vector.

Counter/IV/Nonce Requirements: if packets arrive out of order, the packet must have a sequence number. The mode uses the fill from a primitive stepping polynomial as a pre and post add.

Memory Requirements: marginally more memory is required in software for this mode. Hardware implementations that pipeline lay out the entire stepping of the codebook in hardware.

Ciphertext Expansion: one data block per packet is used for encrypting a checksum that ensures data integrity.