Subliminal Channels: Introduction and Ideas

Lelia Barlow

Abstract— **We introduce the concept of subliminal channels in cryptographic algorithms and protocols. Specific examples are presented, such as subliminal channels in digital signature algorithms. We also present the example of modifications to TCP/IP header fields. Building on these concepts, we explore some new ideas for transmitting hidden messages using an established communication channel.**

I. Introduction

There are two basic ways to hide a message, steganography and cryptography. Steganography conceals the existence of the message. Typically, a secret message is hidden within another message. The method for hiding the secret message is not disclosed to outsiders. This concept has been used throughout history [5]. For example, the second letter of each word in a sentence could spell out a secret message. A secret message could be hidden in a crossword puzzle. The least significant bits of a digital image file could be replaced by the bits of the secret message. The possibilities are endless.

Cryptography does not conceal the existence of the message, but transforms the message using scrambling techniques so that an outsider can not understand the message. Modern techniques include block ciphers such as DES and AES, stream ciphers such as RC4, and public key algorithms such as RSA. Unlike steganographic methods, these cryptographic algorithms are published. The security of the message does not depend on the secrecy of the cryptographic algorithm. Instead, these algorithms use keys for encryption and decryption of messages.

Subliminal channels conceal the existence of a secret message by hiding it within a normal-looking message. However, techniques used in subliminal channels extend beyond simple steganography. Subliminal channels typically require a shared key between the sender and the receiver, and the security of the method depends on the secrecy of this key instead of on the secrecy of the algorithm. Techniques for constructing subliminal channels may be published, without diminishing the effectiveness of the technique.

This intersection between steganography and cryptography is particularly interesting. Not only may secret messages be encrypted using cryptographic algorithms before being communicated in an obfuscated manner via the subliminal channel, but also cryptographic algorithms and protocols may themselves contain subliminal channels. A very specific concern is that subliminal channels in cryptographic algorithms or protocols may "leak" secret information, such as key material.

Author is a student in the Department of Electrical & Computer Engineering, Oregon State University, Corvallis, Oregon 97331. E-mail: `barlow@engr.orst.edu`

A. The Prisoners Problem

In 1983, Gustavus Simmons introduced the concept of a subliminal channel [1]. The concept was based on the following abstract model.

Two accomplices have been arrested for a crime, and will be imprisoned in different cells. The warden permits them to communicate on the condition that the information contained in the messages must be completely open to him. The warden not only wants to be aware of any plans for escape, but also wants to have the opportunity to substitute fabricated or modified messages for genuine messages. In order to plan their escape, the prisoners must find a way to communicate secretly by establishing a subliminal channel. Also, because the prisoners anticipate deception by the warden, they only want to exchange messages that the other may authenticate.

The prisoners solve this problem by subverting the authentication without secrecy channel. If the prisoners have m bits of information to exchange and r bits of authentication, then a total of $(m+r)$ bits must be communicated. Without the knowledge of the warden, the prisoners give up some of their ability to authenticate in exchange for the ability to communicate secretly. To communicate s bits of secret information, the prisoners now have $(r-s)$ bits available for authentication. This means the probability that the warden can succeed in deceiving the prisoners is increased.

Simmons paper describes two examples illustrating this concept. One example also demonstrates that detecting the existence of the subliminal channel could be as difficult as breaking the authentication algorithm.

While Simmons examples [1] were intended only as a proof of concept, it was soon demonstrated that digital signature algorithms could be used to create subliminal channels.

B. Subliminal Channels in Digital Signatures

Both the Ong-Schnorr-Shamir identification scheme [1] and the El Gamal digital signature algorithm [8] can be used to create a subliminal channel [6].

A problem with these schemes is a protocol weakness [2] that allows the subliminal receiver to impersonate the subliminal transmitter. The subliminal transmitter must trust the subliminal receiver with his private key. As a result, the subliminal receiver could forge the digital signature of the subliminal transmitter.

A subliminal channel could also be added to the ESIGN digital signature scheme [2]. This subliminal channel has the advantage that the subliminal receiver can not impersonate the subliminal transmitter. Only a part of the subliminal transmitter's private key is shared with the receiver. It is computationally infeasible for the subliminal receiver to recover the subliminal transmitter's private key.

The Digital Signature Algorithm (DSA) actually has several possibilities for subliminal channels [8], [2]. Each of these possibilities relies on the subliminal transmitter's ability to choose the DSA parameter called k. The parameter k is specified as a 160-bit random number [7].

The simplest subliminal channel in DSA allows the subliminal transmitter to communicate a 160-bit message to the subliminal receiver by choosing a particular value of k. Because the k parameter should appear to be random, the subliminal message should be encrypted using a one-time pad shared between the subliminal transmitter and the subliminal receiver. Another drawback of this technique is that the subliminal transmitter's private key is shared with the subliminal receiver.

DSA also has subliminal channels that do not require the subliminal transmitter's private key to be shared with the subliminal receiver. To send one single bit of subliminal information per signed message, the subliminal transmitter and the subliminal receiver agree on a shared secret key for the subliminal key. This shared secret key is a prime that we will call P (note that this P is not defined by DSA). To communicate the subliminal message "0," the subliminal transmitter chooses a particular value for the k parameter so that the DSA parameter r is a quadratic nonresidue modulo P. To communicate the subliminal message "1," the subliminal transmitter chooses k so that r is a quadratic residue modulo P.

To extend this technique so that multiple subliminal bits are sent in a single signed message, the subliminal transmitter and the subliminal receiver agree on multiple shared secret keys. For example, to send two subliminal bits per message, primes P and Q are established. A value of k is chosen so that r is either a quadratic residue modulo P or a quadratic nonresidue modulo P, and either a quadratic residue modulo Q or a quadratic nonresidue modulo Q.

It is possible that a malicious implementation of DSA can leak bits of the signer's private key [6]. As long as the primes (P, Q, etc.) chosen by the implementation stay a secret, the signer can not prove that her private key was stolen.

These subliminal channels in DSA do not work if the k parameter can not be chosen arbitrarily. One technique proposed [8] is to have the sender and receiver jointly choose k. For example, $k = k'k'' \bmod [p-1]$ where k' is selected by the sender, k'' is selected by the receiver, and p is the DSA parameter. Simmons observed [8] that this technique can allow the receiver to embed a subliminal message in the sender's signature by choosing a particular k'' value. Simmons dubbed this the "Cuckoo's Channel."

C. Anther Type of Hidden Channel

To discuss only the subliminal channel in an authentication without secrecy scenario is to ignore the possibility of hidden channels below the application layer. Information can also be passed from system to system by exploiting the communication channel itself.

A good example is found in the TCP/IP protocol suite[4]. Normal-looking packets can carry secret information if we place this information in certain fields of the TCP/IP header. Although the optional fields may be used for this purpose, required fields are preferable because they are less likely to be altered during transmission.

One required field that may be manipulated is the IP Identification field. The IP Identification field was designed to contain a unique value so that if a packet is fragmented during transmission, it could be correctly re-assembled at the receiver. However, because there were no other requirements placed on the IP Identification field, this 16-bit field is available for use in covert transmissions.

Another interesting field is the 32-bit TCP Initial Sequence Number field. Clearly, an available 32 bits per packet provides more flexibility than 16 bits per packet. However, because tools such as Ethereal (`www.ethereal.com`) can detect out-of-order sequence numbers for a TCP session, this may be a less-desirable technique unless multiple short-lived sessions are used.

D. Applications and Implications

One frequently-cited use of a subliminal channel is the situation where a message is signed under threat. The signer agrees to sign the message, but imbeds the subliminal message that he was coerced. Another possible application involves marking files or digital cash to allow tracking by the entity that produced the mark. Malicious software such as spyware could be used to leak secret information from a host computer without the leak being detected.

It is important to note that the existence of a subliminal channel could compromise the security of a system. Techniques for creating subliminal channels should be studied, in order to better understand possible threats and avenues of attack.

II. New Work

The concept of a subliminal channel is not new, but there have been relatively few publications in this area. The preceding research inspired the author to explore some ideas and potential applications. Although it is difficult to determine whether these ideas are indeed "new," they were not copied from any existing publication.

In the Prisoner's Problem scenario, the sender and receiver are required to communicate in the open. They use an authentication without secrecy channel, and communicate hidden messages by subverting the authentication scheme.

Here, we assume that the sender and receiver are allowed to have secrecy. They may encrypt messages before transmitting on the channel. We further assume that the sender and receiver have unlimited access to the channel and can encrypt whatever content they choose. We explore some possibilities for extending the encryption scheme to include an additional hidden message. Although this may initially appear redundant, we will show that applications for these techniques may exist.

A. Concept

The sender and receiver agree on multiple shared secret keys. For simplicity, we present only the case where two secret keys are shared. Call these keys $Ks1$ and $Ks2$.

In addition, for simplicity, we consider only the case where $Ks1$ and $Ks2$ were generated independently. However, we note that optimizations for an efficient implementation may be possible. For example, if $Ks1$ represents enough key material for 12 rounds of a block cipher, then $Ks2$ could be a truncated version of $Ks1$ that only represents enough key material for 10 rounds of that block cipher. (In this example we assume that 10 or more rounds of the block cipher will provide computational security.)

We encrypt each block of the plaintext message with either $Ks1$ or $Ks2$. If the block was encrypted with $Ks1$, then the receiver interprets the hidden message as a "0" bit. If the block was encrypted with $Ks2$, then the receiver interprets the hidden message as a "1" bit. In this way, each encrypted block contains one hidden bit of information.

Let the plaintext message be represented by M, where $M = M1, M2, M3, \ldots Mk$. Each Mi is a block of length L, and M consists of k blocks. Let the hidden sequence of bits be represented by b, where $b = b1, b2, b3, \ldots bk$. Each bi is a single bit, which could take the value "0" or "1."

In Figure 1, we see that each block of the plaintext message is encrypted with either $Ks1$ or $Ks2$ to produce a corresponding block of ciphertext. The receiver interprets the ciphertext to determine whether $Ks1$ or $Ks2$ was used, and therefore the receiver recovers b.

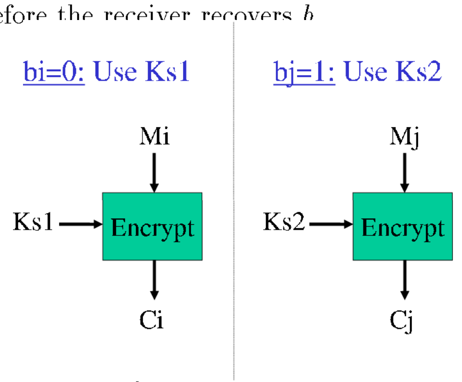

Figure 1: Concept.

It is possible that the original plaintext message could be known to both the sender and receiver prior to the communication. However, this implies that the message is analogous to a lengthy version of a one-time pad. Instead, we focus on techniques to recover the message as well as the hidden sequence of bits.

To recover the message, we need a way to determine if decryption is successful with a given key. In other words, the decrypted message must be recognizable to the receiver decrypting it. We will assume this is the case, and that if overhead is incurred, it is incurred whether this concept is utilized or not.

B. Variations

Several variations on this basic concept are possible. Three variations are described below.

Variation A

The most basic methodology is described in Figure 2. The sender encrypts a message and sends each block of ciphertext, here called Ci, to the receiver. The receiver decrypts Ci using $Ks1$ to produce $Mi*$. Then the receiver tests whether $Mi*$ is correct: $Mi* = Mi$. If so, then Mi was encrypted using $Ks1$ and $bi = 0$. If not, then Mi was encrypted using $Ks2$ and $bi = 1$. To recover Mi, the receiver muse decrypt again, this time using $Ks2$. Note that, on average, the receiver will decrypt 1.5 times for each block received.

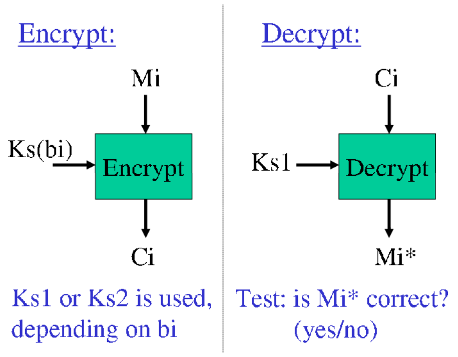
Encrypt:
Mi
Ks(bi)
Encrypt
Ci
Ks1 or Ks2 is used, depending on bi
Decrypt:
Ci
Ks1
Decrypt
Mi*
Test: is Mi* correct? (yes/no)

Figure 2: Variation A.

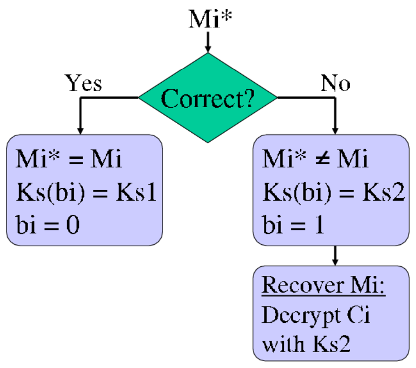

Figure 3 shows the steps involved in the test to determine whether $Mi*$ is correct. If $Mi*$ is not correct, then a second decryption is necessary to recover Mi.

Variation B

Figure 4 describes an alternate approach. Here, there is no plaintext message to send. A counter is incremented, and used in place of the plaintext blocks. The sender encrypts the counter value using the key determined by the value of bi, and produces a result called Vi. Note that Vi is of length L.

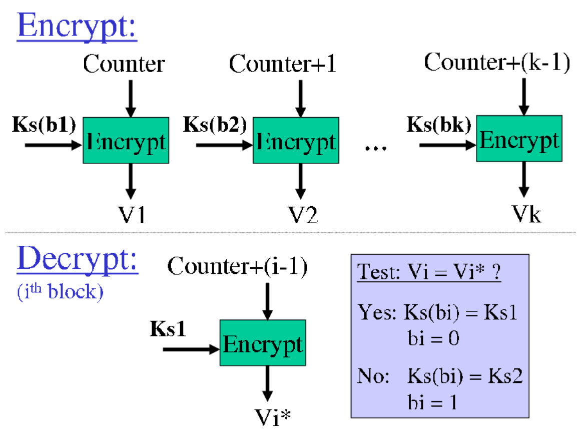

Figure 4: Variation B.

Note that there is no need to recover a message, and therefore only a single encryption is needed to decode the sequence b.

Variation C: CTR Mode

In Figure 5, we present an approach that is very similar to the Counter Mode of Operation for block ciphers. An advantage of this method is that plaintext statistics would be difficult to exploit. Further, it has the advantage that both Vi values (that is, one Vi value for $Ks1$ and one Vi value for $Ks2$) can be pre-computed by the receiver before Ri arrives.

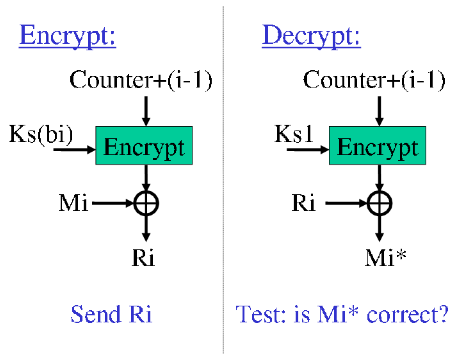

Figure 5: Variation C.

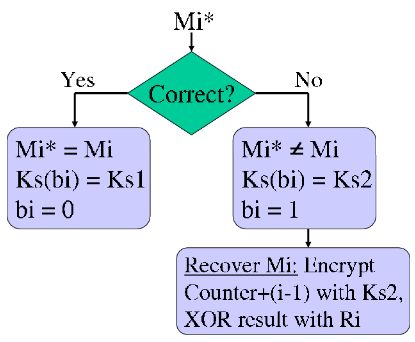

Figure 6: Decrypt Test.

Figure 6 shows the steps involved in the test to determine whether $Mi*$ is correct. If $Mi*$ is not correct, then a second encryption is necessary to recover Mi.

C. Possible Applications

A variety of applications of this concept can be imagined. Four possible applications are described: expanding a sequence, stamping a message, restricting access to portions of a message, and reducing session rekeying overhead.

Sequence Expansion

Variation B may be used to expand a k-bit sequence into a sequence of length kxL. (Recall that L is the block length.) Here, let $b = b1, b2, b3, \ldots bk$ be the binary sequence. The expanded sequence can be expressed:

$$\begin{aligned} Sequence = Encrypt[Ks(b1), counter], \\ Encrypt[Ks(b2), counter + 1], \ldots \\ Encrypt[Ks(bk), counter + (k-1)] \end{aligned}$$

The expanded sequence can be re-generated if b, $counter$, $Ks1$, and $Ks2$ are known. However, the expanded sequence would be computationally difficult to create without $Ks1$ and $Ks2$. Conversely, it would be difficult to determine b without $Ks1$ and $Ks2$.

Stamp a Message

Either Variation A or Variation C could be used to “stamp” a message. Figure 7 describes this approach.

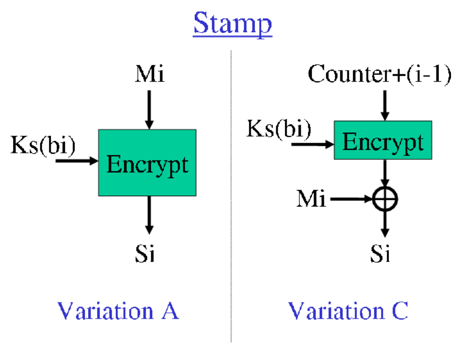

Figure 7: Stamp a Message.

The Si values form the basis of the stamp. There are several possibilities for forming the stamp from the sequence of Si values:

- Stamp = $S1, S2, S3, \ldots Sk$
- Stamp = Hash $(S1, S2, S3, \ldots Sk)$
- Stamp = HMAC $(S1, S2, S3, \ldots Sk)$
- Stamp = DAC $(S1, S2, S3, \ldots Sk)$

This concept of a stamp differs from the concept of a digital signature. A digital signature provides non-repudiation because only the signer knows the private key. Also, a digital signature can be verified by anyone who has the corresponding public key.

In contrast, only authorized individuals share $Ks1$ and $Ks2$. Only authorized individuals can stamp a message, and verify a stamp on a message. Therefore, the stamp only has significance within this group of authorized individuals. Further, each authorized individual could produce a different but verifiable stamp using a different sequence b. These sequences could be assigned by the group. If the sequence b has length k, then up to 2^k group members can be accommodated in this scheme.

To verify a stamp, M, b, $Ks1$, and $Ks2$ are needed. A caution of this method is that authorized individuals can impersonate each other. Once I verify your stamp, I know your sequence b. In addition, to avoid replay attacks, a timestamp or nonce should be added to the message before it is stamped.

Restrict Access to Portions of a Message

Suppose that certain information in a message M is designated "confidential." Only authorized individuals should be allowed to view these parts of the message. Two versions of the message could be created: one for individuals with authorization, and one for individuals without authorization. Alternatively, a single version of the message could be sent to all individuals, if confidential information was hidden from unauthorized individuals.

Suppose the message M consists of five blocks: $M = M1, M2, M3, M4, M5$. Blocks $M2$ and $M5$ contain confidential information. Therefore, the sequence b is chosen such that $b = b1, b2, b3, b4, b5$. The value 0 is assigned to $b1$, $b3$, and $b4$. The value 1 is assigned to $b2$ and $b5$. Now either Variation A or Variation C could be used to encrypt the entire message. Unauthorized recipients have $Ks1$, and can recover only blocks $M1$, $M3$, and $M4$. Authorized recipients have $Ks1$ and $Ks2$, and can recover the entire message.

Note that if less than a full block of information is confidential, the block could be padded to the correct length before encrypting. Similarly, if there is less than a full block of information that is not confidential, it is padded to the correct length before encrypting.

Reduce Session Rekeying Overhead

Variation A or Variation C could be used to transmit "raw" session key material between the sender and the receiver while encrypted messages are exchanged. This application allows the session key to be updated without the overhead of additional protocol messages. The tradeoff is that the sender and receiver have more processing work to do. This method is described in Figure 8.

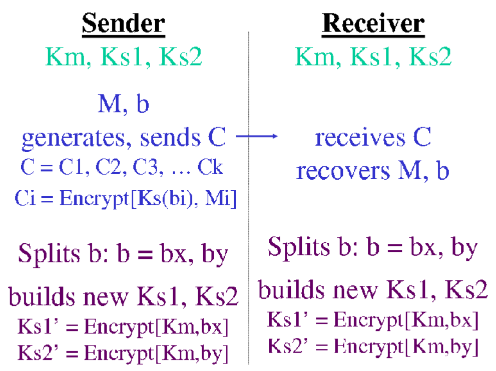

Figure 8: Session Rekey.

The sender and receiver share a master key, Km, and a split session key, $Ks1$ and $Ks2$. The sender has a message M and a recently-generated sequence b, and produces the ciphertext C. The ciphertext C is sent to the receiver. The receiver obtains C, and recovers M and b. The sender and the receiver split the sequence b in half, so that the first half of b is called bx and the second half of b is called by. Thus, $b = bx, by$. Note that the sequence b has an even number of terms; in other words, k is even. The new session key is computed as:

$$Ks1_new = Encrypt[Km, bx]$$
$$Ks2_new = Encrypt[Km, by].$$

The values bx and by are encrypted with the master key to obtain forward secrecy of the session keys. Without this step, if an adversary knows one session key, then the adversary can determine all future session keys.

III. Future Work

Any of the above ideas could be further refined and expanded. Simulation may help to uncover any potential weaknesses the author has not yet imagined.

References

[1] G. J. Simmons, "The prisoner's problem and the subliminal channel," *Advances in Cryptology: Proceedings of CRYPTO '83*, Plenum Press, 1984, pp.51–67.
[2] G. J. Simmons, "A secure subliminal channel (?)," *Advances in Cryptology — CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 33-41.
[3] Y. Desmedt, "Abuses in cryptography and how to fight them," *Advances in Cryptology — CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 375-389.
[4] C. Rowland, "Covert channels in the TCP/IP protocol suite," *First Monday*, web address, 2005.
[5] D. Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Macmillan Publishing Co., 1967.
[6] B. Schneier, *Applied Cryptography Second Edition: Protocols, Algorithms, and Source Code in C*, New York: John Wiley and Sons, Inc., 1996.
[7] National Institute of Standards and Technology, "Digital Signature Standard," *NIST FIPS PUB 186,*, U.S. Department of Commerce, May 1994.
[8] G. J. Simmons, "Subliminal channels: past and present," *European Transactions on Telecommunications*, vol. 4, no. 4, Jul. Aug. 1994, pp. 459–473.
[9] G. J. Simmons, "Subliminal communication is easy using the DSA," *Advances in Cryptology — EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 218-232.
[10] Y. Desmedt, "Subliminal-free authentication and signature," *Advances in Cryptology — Eurocrypt '88 Proceedings*, Springer-Verlag, 1988, pp. 23-33.