

Defending DSSS-based Broadcast Communication against Insider Jammers via Delayed Seed-Disclosure

Abstract

Spread spectrum techniques such as Direct Sequence Spread Spectrum (DSSS) and Frequency Hopping (FH) have been commonly used for anti-jamming wireless communication. However, traditional spread spectrum techniques require that sender and receivers share a common secret in order to agree upon, for example, a common hopping sequence (in FH) or a common spreading code sequence (in DSSS). Such a requirement prevents these techniques from being effective for anti-jamming *broadcast* communication, where a jammer may learn the key from a compromised receiver and then disrupt the wireless communication. In this paper, we develop a novel Delayed Seed-Disclosure DSSS (DSD-DSSS) scheme for efficient anti-jamming broadcast communication. DSD-DSSS achieves its anti-jamming capability through randomly generating the spreading code sequence for each message using a random seed and delaying the disclosure of the seed at the end of the message. We also develop an effective protection mechanism for seed disclosure using content-based code subset selection. DSD-DSSS is superior to all previous attempts for anti-jamming spread spectrum broadcast communication without shared keys. In particular, even if a jammer possesses real-time online analysis capability to launch reactive jamming attacks, DSD-DSSS can still defeat the jamming attacks with a very high probability. We evaluate DSD-DSSS through both theoretical analysis and a prototype implementation based on GNU Radio; our evaluation results demonstrate that DSD-DSSS is practical and have superior security properties.

1 Introduction

Spread spectrum wireless communication techniques, including Direct Sequence Spread Spectrum (DSSS) and Frequency Hopping (FH), have been commonly used for anti-jamming wireless communication [6]. However, with traditional spread spectrum techniques, it is necessary for senders and receivers to share a secret key to determine, for example, the frequency hopping patterns in FH and the Pseudo-Noise (PN) codes in DSSS. Otherwise, sender and receivers cannot establish anti-jamming communication. Moreover, if a jammer knows the secret key, she can also replicate the secret hopping pattern or PN codes and jam the wireless communication.

The above limitations of traditional anti-jamming techniques have motivated a series of recent research. To remove the dependency on pre-shared keys, an Uncoordinated Frequency Hopping (UFH) technique was recently developed to allow two nodes to establish a common secret for future FH communication in presence of a jammer [17]. This approach was latter enhanced in [7, 16, 18] with various coding techniques to provide more efficiency and robustness during key establishment. Unfortunately, UFH and its variations [7, 16–18] are limited to point-to-point communication, and cannot be extended to broadcast communication.

To address this problem, two approaches were recently investigated to enable jamming-resistant broadcast communication *without* shared keys [2, 13]. BBC was proposed to achieve broadcast communication by encoding data into “indelible marks” (e.g., short pulses) placed in “locations” (e.g., time slots), which can be decoded by any receiver [2, 3]. However, the decoding process in BBC is inherently sequential (i.e., the decoding of the next bit depends on the decoded values of the previous bits). Though it works with short pulses in the time domain, the method cannot be extended to DSSS or FH without significantly increasing the decoding cost. An Uncoordinated DSSS (UDSSS) approach was recently developed [13], which avoids jamming by randomly selecting the spreading code sequence for each message from a public pool of code sequences. UDSSS allows a receiver to quickly identify the right code sequence by having each code sequence uniquely identified by the first

few codes. However, if the jammer has enough computational power, using the same property, she can find the correct sequence before the sender finishes the transmission and jam the remaining transmission. Thus, UDSSS is vulnerable to *reactive jamming attacks*, where the jammer can analyze the first part of transmitted signal and jam the rest accordingly.

In this paper, we develop Delayed Seed-Disclosure DSSS (DSD-DSSS), which provides efficient and robust anti-jamming broadcast communication without suffering from reactive jamming attacks. The basic idea is two-fold: First, the code sequence used to spread each message is randomly generated based on a random seed only known to the sender. Second, the sender discloses the random seed at the end of the message, after the main message body has been transmitted. A receiver buffers the received message; it can decode the random seed and regenerate the spreading code using the seed to despread the buffered message. A jammer may certainly attempt the same thing. However, when the jammer recovers the random seed and the spreading code sequence, all reachable receivers have already received the message; it is too late for the jammer to do any damage.

We also develop a *content-based code subset selection* scheme to protect the random seed disclosure. We use the content of the seed to give some advantage to normal receivers over reactive jammers. This scheme allows a normal receiver, who starts decoding a message after fully receiving the message, to quickly decode the random seed. In contrast, a jammer, who needs to disrupt the message while it is being transmitted, has to consider many more choices.

Our contribution in this paper is as follows. First, we develop the novel DSD-DSSS scheme to provide efficient anti-jamming broadcast communication without shared keys. Our approach is superior to all previous solutions. Second, we develop a content-based code subset selection method to provide effective protection of seed disclosure in DSD-DSSS. Third, we give in-depth performance and security analysis for these techniques in presence of various forms of jammers, including reactive jammers that possess real-time online analysis capabilities. Our analysis demonstrates that our approach provides effective defense against jamming attacks. Finally, we implement a prototype of DSD-DSSS using USRPs and GNU Radio to demonstrate its feasibility.

The remainder of the paper is organized as follows. Section 2 describes background information about DSSS. Section 3 presents our assumptions and the threat model. Section 4 proposes DSD-DSSS and analyzes its anti-jamming capability and performance overheads. Section 5 gives the content-based code subset selection scheme and analyzes its effectiveness. Section 6 shows the implementation and experimental evaluation of DSD-DSSS. Section 7 describes related work, and Section 8 concludes this paper.

2 Background

Spread spectrum techniques, including DSSS and FH, use a much larger bandwidth than necessary for communications [6, 14]. Such bandwidth expansion is realized through a spreading code *independent* of the data sequence. In DSSS, each data bit is spread (multiplied) by a wide-band code sequence (i.e., the *chipping sequence*). The spreading code is typically pseudo-random, commonly referred to as *Pseudo-Noise (PN) code*, rendering the transmitted signal noise-like to all except for the intended receivers, which possess the code to despread the signal and recover the information.

Figure 1 shows the typical steps in DSSS communication. Given a message to be transmitted, typically encoded with Error Correction Code (ECC), the sender first spreads the message by multiplying it with a spreading code. Each bit in the message is then converted to a sequence of chips¹ according to the spreading code. The result is modulated, up-converted to the carrier frequency, and launched on the channel. At the receiver, the distorted signal is first down-converted to baseband, demodulated through a matched filter, and then despread by a synchronized copy of the spreading code. The synchronization includes both bit time synchronization and chip time synchronization, guaranteeing that receivers know when to apply which spreading code in order to get the original data. Alternatively, a DSSS system may modulate the signal before the spreading step at sender, and despread and demodulate the received signal at receiver.

¹To distinguish between bits in the original message and those in the spreading result, following the convention of spread spectrum communication, we call the “shorter bits” in the spreading result as *chips*.

The performance of DSSS communication depends on the design of spreading codes. A spreading code $c(t)$ typically consists of a sequence of l chips c_1, c_2, \dots, c_l , each with value 1 or -1 and duration of T_c , where l is the code length and T_c is chip duration. Assume the bit duration is T_b . The number of chips per bit $l = T_b/T_c$ well approximates the bandwidth expansion factor and the processing gain. Two functions characterize spread code: *auto-correlation* and *cross-correlation*. Auto-correlation describes the similarity between a code and its shifted value. Good auto-correlation property means the similarity between a code and its shifted value is low; it is desired for multi-path rejection and synchronization. Cross-correlation of two spreading codes describes the similarity between these two codes; low cross-correlation is desired for multiuser communications.

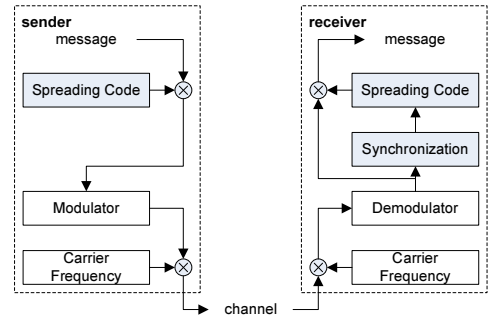


Figure 1: DSSS communication system

3 Assumptions and Threat Model

In this paper, we consider the protection of DSSS-based wireless *broadcast* communication against jamming attacks (i.e., one sender and multiple receivers). We adopt the same DSSS communication framework as illustrated in Figure 1. However, the sender and receivers use different strategies to decide what spreading codes to use during broadcast communication. That is, our approach customizes the generation and selection of spreading codes during DSSS communication to defend against insider jamming attacks.

We assume that the jammers’ transmission power is bounded. In other words, a jammer cannot jam the transmission of a message unless she knows the spreading codes used for sending the message. For simplicity, we assume the length of each broadcast message is fixed. Such an assumption can be easily removed, for example, by using a message length field.

Threat Model: We assume that the attacker may compromise some receivers, and as a result, can exploit any secret they possess to jam the communication from the sender to the other receivers. We assume intelligent jammers that are aware of our schemes. In addition to injecting random noises, the jammer may also modify or inject meaningful messages to disrupt the broadcast communication.

The jammers may possess high computational capability to perform real-time online analysis of intercepted signal. However, due to the nature of DSSS communication (i.e., each bit data is transmitted through a sequence of pseudo-random chips), it takes time for a jammer to parse the chips for any 1-bit data to determine the spreading code. When the jammer receives enough chips for a given bit to guess the spreading code with a high probability, most of the chips have already been transmitted. Jamming the remaining chips will not have high impact on the reception of this bit. Thus, we assume that if a jammer does not know the spreading code for any 1-bit data, she cannot jam its transmission based on real-time analysis of the signal.

4 Basic DSD-DSSS

The basic idea of DSD-DSSS is two-fold. First, the code sequence used to spread a broadcast message is randomly generated based on a random seed only known to the sender. Thus, nobody except for the sender knows the right spreading code sequence before the sender discloses it. Second, the sender discloses the random seed at the end of the broadcast message, after the main message body has been transmitted. A receiver buffers received signal (or more precisely, received chips); it can decode the random seed and regenerate the spreading code sequence accordingly to despread the buffered chips. A jammer may certainly attempt the same thing. However, when the jammer recovers the seed and the spreading code sequence, all reachable receivers have already received the message. It is too late for the jammer to do any damage. Figure 2 illustrates the sending and receiving processes in DSD-DSSS. In the following, we describe this new scheme in detail.

4.1 Spreading Code Sets

Similar to traditional DSSS communication, DSD-DSSS uses spreading codes with good auto-correlation and low cross-correlation properties (e.g., PN codes).

DSD-DSSS keeps two sets of *publicly known* spreading codes: C_p and C_e . Codes in C_p are used to spread the message body m_i , while codes in C_e are used to spread the random seed at the end of each message. We require that C_p and C_e have no overlap (i.e., $C_p \cap C_e = \emptyset$). For convenience, we give each code in C_p (or C_e) a unique index. For a given index i for C_p (or C_e), we use $C_p[i]$ (or $C_e[i]$) to refer to the i -th code in C_p (or C_e).

We use individual bits in the message as the basic units of spreading. That is, each bit is spread with a different spreading code. As a result, even if an intelligent jammer can infer the spreading code for the current bit through real-time analysis, she cannot use this code to jam the following bit.

4.2 Sender

Given a l_m -bit message m_i , the sender encodes m_i in two parts: *message body* and *random seed*.

Spreading Message Body: The sender first generates a random seed s_i , and then uses a pseudo-random generator with seed s_i to generate a sequence of l_m random indexes $mid_1 || mid_2 || \dots || mid_{l_m}$, where $1 \leq mid_i \leq |C_p|$. The sender then generates a sequence of spreading codes cs_m for m_i by drawing codes from C_p using these indexes. That is, $cs_m = C_p[mid_1] || C_p[mid_2] || \dots || C_p[mid_{l_m}]$. The sender then uses cs_m to spread m_i (i.e., each code $C_p[mid_k]$ is used to spread the k -th bit of m_i). For convenience, we denote the spread message body (more precisely, the spread chips) as $S(cs_m, m_i)$.

Spreading Seed: A naive method is to disclose the seed s_i right after the spread message body $S(cs_m, m_i)$ so that receivers can recover s_i from the end of the message, generate cs_m using s_i , and despread the message. However, such a method is highly vulnerable to jamming attacks. Indeed, a jammer can simply disrupt the transmission of the seed to prevent the message from being received.

To prevent jamming attacks against the disclosed seed, the sender spreads the seed s_i using codes randomly selected from C_e , one of the public code sets. Assume the seed has l_s bits. The sender randomly draws l_s codes independently from C_e to form a sequence of l_s spreading codes, denoted $cs_s = C_e[sid_1] || \dots || C_e[sid_{l_s}]$, where sid_1, \dots, sid_{l_s} are random integers between 1 and l_s . The sender then spreads the k -th bit in the seed s_i with the corresponding code $C_e[sid_k]$, where $1 \leq k \leq l_s$. The spreading results are then modulated, up-converted to the carrier frequency, and transmitted in the communication channel.

4.3 Receiver

As shown in Figure 2, each receiver keeps sampling the channel through down-conversion and demodulation, and saves the received chips in a cyclic buffer. Each receiver continuously processes the buffered chips to recover possibly received messages. To recover a meaningful message, a receiver has to first synchronize the buffered chips (i.e., align the buffered chips with appropriate spreading code) and then despread them.

Synchronization and Recovery of Seed: The goal of synchronization is to identify the positions of the chips of a complete message in the buffer before despreading them. The key for synchronization is to locate the seed, which occupies the last $l \times l_s$ chips in a message.

As shown in Figure 2, a receiver uses a sliding window with window size $l_s \times l$ to scan and locate the seed in the buffer, where l_s is the number of bits in a seed and l is the number of chips in a spreading code. The sliding window is shifted to the right by 1 chip each time.

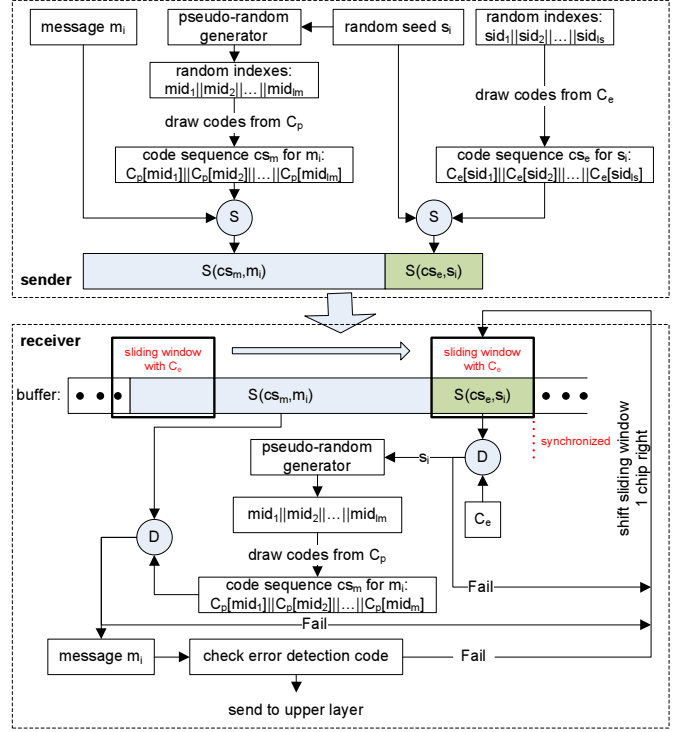


Figure 2: Delayed seed-disclosure DSSS (DSD-DSSS)

In each scan, the receiver first uses the public code set C_e to despread the chips in the sliding window to synchronize with the sender. Conceptually, the receiver partitions the $l_s \times l$ chips into l_s groups, and tries each code in C_e to despread each group in the window. Note that using a set of codes with good auto-correlation and low cross-correlation properties, we can get high correlation and despread a bit successfully only when the same code (as the one used for spreading) is used to despread the encoded chips in the right position. If the despreading is successful for every group, the content in the window is a seed, which has been successfully recovered. At the same time, the position of the message body in the buffer is determined, i.e., the $l_m \times l$ chips to the left of the window in the buffer belong to the message body. Otherwise, the receiver shifts the window to the right by 1 chip and repeats the same process. This process can certainly be further optimized. We omit the details, since it is not critical for the presentation of our approach.

Despreading Message Body: Once a receiver recovers a seed s_i and determines the position of a received message in the buffer, it follows the same procedure as the sender to generate the sequence of spreading codes $cs_m = C_p[mid_1] || C_p[mid_2] || \dots || C_p[mid_{l_m}]$. The receiver then despreads the message body using cs_m . Specifically, the receiver partitions the chips buffered for the message body into l_m groups, each of which has l chips, and uses code $C_p[mid_k]$ to despread the k -th group of chips ($1 \leq k \leq l_m$).

At the end of this process, the receiver will recover the message body m_i and forward it to upper-layer protocols for further processing (e.g., error detection, signature verification).

4.4 Security Analysis

To show the effectiveness of DSD-DSSS against jamming attacks, we analyze the jamming probability (i.e., the probability that the message is jammed) in DSD-DSSS under different jamming attacks. Following the classification in [12], we consider two kinds of jamming attacks: *non-reactive* jamming and *reactive* jamming attacks. A non-reactive jammer continuously jams the communication channel without knowledge about the message transmission. Reactive jammer detects the transmission before jamming the channel. The jammer can apply three strategies to each attack: *static*, *sweep*, and *random* strategies. In the static strategy, the jammer uses the same code to jam the channel all the time. In the sweep strategy, the jammer periodically changes the code for jamming and does not reuse a code until all other codes have been used. In the random strategy, the jammer periodically changes the jamming code to a random code.

We also consider Denial of Service (DoS) attacks targeting at seed disclosure at receivers, in which the jammer attempts to force receivers to deal with a large number of candidate seeds.

4.4.1 Jamming Attacks

DSD-DSSS provides strong resistance against jamming attacks. Because each message is spread with a pseudo-random code sequence decided by a random seed, no one except for the sender can predict the spreading code sequence and jam the communication. The random seed is disclosed at the end of each message. Thus, when a jammer learns the seed, it is already too late to jam the transmitted message with it. A jammer may certainly try to jam the transmission of the random seed. However, each bit of the seed is spread with a code randomly selected from a code set (i.e., C_e), making it hard for a jammer to predict.

In the following, we provide a quantitative analysis of the jamming probabilities in various jamming scenarios. A jammer has two targets in each message: message body and seed. The jammer may jam the message body directly, or alternatively, the seed so that receivers cannot recover the seed and then the spreading code sequence for the message body. To successfully jam even one bit of the message body, the jammer has to know the spreading code for that bit and precisely synchronize its chips with those of the transmitted message.

Non-reactive Jamming Attacks: Non-reactive jammers do not rely on any information about the transmitted messages. Thus, they have to guess the spreading code and synchronization. We consider all three jamming strategies (i.e., static, sweep, and random strategies) [12] and provide the jamming probabilities in the following two Theorems. The proofs are trivial and omitted due to space limit.

Theorem 1. *When DSD-DSSS is used, the jamming probability of a non-reactive jammer with the static strategy is at most $1 - \left(1 - \frac{1}{|C_p|}\right)^{l_m}$ if the jammer targets the message body, and is at most $1 - \left(1 - \frac{1}{|C_e|}\right)^{l_s}$ if the*

jammer targets the seed.

Theorem 2. *When DSD-DSSS is used, the jamming probability of a non-reactive jammer with the random (or sweep) strategy is at most $1 - \left(1 - \frac{1}{l(|C_p|+|C_e|)}\right)^{l_m+l_s}$.*

Reactive Jamming Attacks: A reactive jammer can detect the sender’s transmission and perform real-time analysis of the transmitted signal. It can further synchronize with the sender so that she knows the precise chip layout of the transmitted message. However, as mentioned in Section 3, if a reactive jammer does not know the spreading code for any given bit data, she cannot jam the transmission based on real-time analysis. Nevertheless, the reactive jammer only needs to guess the sender’s spreading code to jam the communication. This increases the jamming probability compared with simple non-reactive jamming attacks. Similar to non-reactive jammer, the reactive jammer can also use static, random, or sweep jamming strategies to jam the channel. We give the jamming probability for all three strategies in Theorem 3 below. (The proof is omitted due to space limit.) Note that the jamming strategy no longer has direct impact on the maximum jamming probability.

Theorem 3. *When DSD-DSSS is used, the jamming probability of reactive jamming attacks is at most $1 - \left(1 - \frac{1}{|C_p|}\right)^{l_m} \cdot \left(1 - \frac{1}{|C_e|}\right)^{l_s}$.*

Figure 3 shows the jamming probabilities of both non-reactive and reactive jamming attacks, in which $|C_p| = |C_e|$, both ranging from 1,000 to 7,000, the sizes of message body and random seed are $l_m = 1,024$ bits and $l_s = 64$ bits, respectively, and the length l of each code is set to 100 or 200. Figure 3 shows that the reactive jamming attacks have much more impact than non-reactive jamming attacks due to the jammer’s ability to synchronize with the sender. In all non-reactive jamming attacks, the jamming probabilities are no more than 0.01. However, even when $|C_p| = |C_e| = 7,000$, the reactive jammer’s jamming probability is still 0.14. Figure 3 also shows that using Error Correction Code (ECC) can reduce the jamming probability dramatically. Simply using an ECC that can tolerate 1 bit error can lower the reactive jammer’s jamming probability from 0.14 to 0.009.

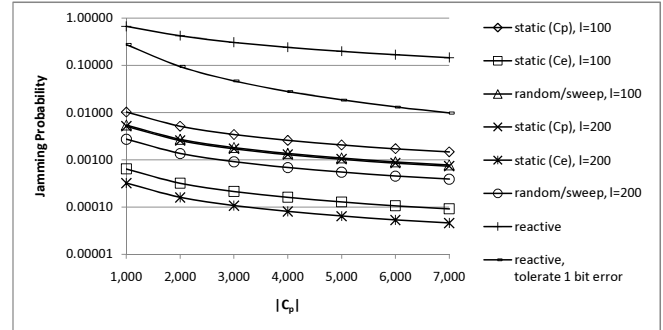


Figure 3: Maximum jamming probability for non-reactive and reactive jamming attacks ($l_m = 1024$; $l_s = 64$; $|C_e| = |C_p|$; $l = 100$ or 200)

The above results demonstrate that DSD-DSSS is effective in defending against jamming attacks, even when the jammer launches sophisticated reactive jamming attacks.

4.4.2 DoS Attacks against Seed Disclosure

DSD-DSSS has good resistance against various jamming attacks. However, an attacker may also inject bogus seeds or bogus messages, faking message transmissions from the sender. Indeed, this is a problem common to all wireless communication systems. As long as a communication channel is accessible to an attacker, she can always inject fake messages. An authentication mechanism (e.g., digital signature) is necessary to filter out such fake messages.

An attacker may go one step further to launch DoS attacks targeting the seed disclosed at the end of each message. Specifically, the attacker may inject bogus seeds by continuously drawing a code from C_e , spreading a random bit, and transmitting it to receivers. A receiver will see a continuous stream of possible seeds being disclosed. Without any further protection, the receiver will have to attempt the decoding of a message with all possible seeds. An attacker may use multiple transmitters to inject multiple transmissions of each bit in a seed. As a result, the receiver may have to try the combinations of these options when decoding the messages. In Section 5, we will present an enhanced scheme to better protect seed disclosure against such DoS attacks in DSD-DSSS.

4.5 Performance Overheads

Computation Overhead and Delay: In terms of computation, the sender needs to generate a random seed, generate a spreading code sequence using a pseudo-random generator, and spread both the seed and the message body. All these operations can be performed efficiently and lead to negligible delay.

A receiver needs to synchronize with the sender’s chips, despread and decode the seed, regenerate the spreading code sequence for the message body, and despread the message body. With the exception of synchronization and recovery of the seed, all other operations can be efficiently performed. Synchronization and recovery of seed are computationally expensive. A receiver should use all codes in C_e to despread every l chips in the buffer. Compared with traditional DSSS, this process is at least $|C_e|$ times more expensive.

DSD-DSSS introduces more receiver side delay than traditional DSSS, particularly because a receiver cannot start decoding a received message until the seed is recovered. Assume a straightforward implementation on the receiver side. For a received message, the time delay for the receiver to find the seed is $l(l_m + 1)|C_e|t$, and the time delay to further recover the seed is $(l_s - 1)|C_e|t$, where t is the time required to despread l chips. The sum of these two delays constitute the majority of the receiver side delay. Note that this process can be parallelized to reduce the receiver side delay.

Storage Overhead: DSD-DSSS requires a buffer to store the chips of a potential incoming message. When a message is being processed, a receiver has to buffer another message potentially being transmitted. Moreover, when there are multiple senders broadcasting at the same time, a receiver needs to buffer for decoded messages from all of them. Thus, in DSD-DSSS, a receiver needs storage that is possibly tens of times of what is required by traditional DSSS. Nevertheless, considering the typical message size (e.g., a few hundred bytes) and the low cost of memory nowadays, such a storage overhead is certainly affordable on a communication device.

Communication Overhead: DSD-DSSS adds a random seed at the end of each broadcast message, resulting in more communication overhead than traditional DSSS. Nevertheless, compared with the size of a typical message body (e.g., a few hundred bytes), the size of a random seed (e.g., 8 bytes) is negligible. Thus, DSD-DSSS introduces very light communication overhead.

5 Efficient and Jamming-resistant Seed Disclosure

In this section, we enhance the basic DSD-DSSS scheme by developing a more effective protection of seed disclosure for the DoS threat discussed in Section 4.4.2. This approach gives normal receivers more advantages over jammers. It is based on the observation that a normal receiver can wait until a message is fully received to decode its content, while a jammer, to be effective in jamming, has to determine the jamming code when the message is being transmitted.

We propose *content-based code subset selection* for spreading and despreading the seed. The basic idea is to use the content of the seed to give some advantage to normal receivers. Specifically, the sender spreads the seed bit-by-bit from the end to the beginning. For each bit (except for the last one), the sender uses both the value and the spreading code of the later bit to determine its candidate spreading codes, which are a small subset of all possible codes. Note that when a receiver starts decoding a message, it already has the entire message buffered. Thus, a receiver can follow the same procedure as the sender to recover the small subset of candidate codes for each bit of the seed. However, without the complete message, a jammer has to consider many more spreading codes. Any code not in the right subset will be ignored by normal receivers. Moreover, even if some codes chosen by jammers are accepted by chance, the receivers do not need to consider the combinations of all accepted codes in different bit positions in the seed, avoiding the most serious DoS attack.

Recall that the basic DSD-DSSS scheme employs two public code sets C_p and C_e , where only C_e is used to spread the seed. In the new approach, we enhance the protection of the seed by using both code sets. The codes in C_e are only used to spread the last bit of the seed, marking the end of the seed. We generate multiple subsets of C_p . Each earlier bit of the seed is spread with one of these subsets, selected based on the value and spreading code of the later bit.

A reactive jammer may attempt to infer the code used to spread the next bit based on her current observation (i.e., the code used for the current bit). It is critical not to give the jammer such an opportunity. Thus, we require

that each code appear in multiple subsets of C_p . As a result, knowing the code for the current and past bits does not give any jammer enough information to make inference for future bits.

5.1 Generation of Subsets of C_p

To meet the requirement for the subsets of C_p , as a convenient starting point, we choose *finite projective plane*, which is a symmetric Balanced Incomplete Block Design (BIBD) [8], to organize the spreading codes in C_p . It is certainly possible to use other combinatorial design methods to get better properties. We consider these as possible future work, but do not investigate them in this paper.

A *finite projective plane* has $n^2 + n + 1$ points, where n is an integer called the *order* of the projective plane [8]. It has $n^2 + n + 1$ lines, with $n + 1$ points on every line, $n + 1$ lines passing through every point, and every two points appearing together on exactly 1 line. It is shown in [8] that when n is a power of a prime number, there always exists a finite projective plane of order n .

In this paper, we consider the points on a finite projective plane as spreading codes in C_p and lines as subsets of C_p . For a finite projective plane with order n , we associate each point with a spreading code and each line with a subset. We construct C_p by selecting $n^2 + n + 1$ spreading codes with good auto-correlation and low cross-correlation properties (e.g., PN codes [6]). As a result, we also have $n^2 + n + 1$ subsets, where each subset has $n + 1$ codes, each code appears in $n + 1$ subsets, and every two codes co-exist in exactly 1 subset. We give a unique index to each subset of C_p to facilitate the selection of subsets during spreading and despreading.

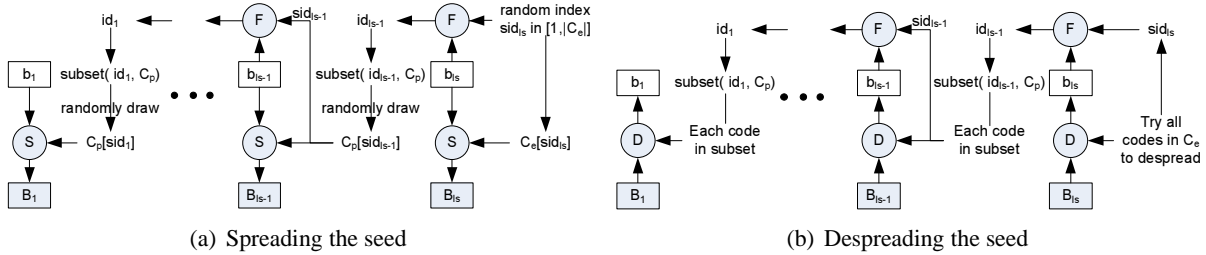


Figure 4: Content-based code subset selection

5.2 Spreading the Seed

Figure 4(a) shows how the sender spreads the seed. We represent each bit of the seed as b_i , where $1 \leq i \leq l_s$ and l_s is the number of bits in the seed. As mentioned earlier, the sender spreads the seed from the end to the beginning.

For bit b_{l_s} , the sender randomly chooses a code from C_e and spreads b_{l_s} with this code to get a sequence of chips B_{l_s} . Assume the index of the chosen code is sid_{l_s} , where $1 \leq sid_{l_s} \leq |C_e|$.

We use a function F to determine which subset of C_p is used for the next (earlier) bit. Function F has two inputs: the index of a code in C_p or C_e , and a bit value (1 or 0). The output of F is the index of a subset of C_p . F can be any function that reaches the indexes of the subsets of C_p evenly with evenly distributed inputs. To guarantee that any subset of C_p be used for b_{l_s-1} , we must have $|C_e| \geq \left\lceil \frac{|C_p|}{2} \right\rceil$. For simplicity, we set $|C_e| = \left\lceil \frac{|C_p|}{2} \right\rceil$. Specifically, for bit b_i , where $1 \leq i \leq l_s - 1$, the sender uses sid_{i+1} and b_{i+1} as the input of F to get id_i , the index of subset for bit b_i . The sender then randomly draws a code from the subset of C_p with index id_i to spread bit b_i and get the sequence of chips B_i . Assume that the code's index is sid_i . The sender continues this process to spread the earlier bits.

5.3 Despreading the Seed

Figure 4(b) shows how a receiver despreads the seed. The receiver continuously tries to find the end of a message in the buffer using a sliding window method as discussed in Section 4.

In the sliding window, the receiver sequentially tries every code in C_e to despread the last l chips in the window. If no code in C_e can successfully despread the last l chips, the sliding window shifts 1 chip to the right in the

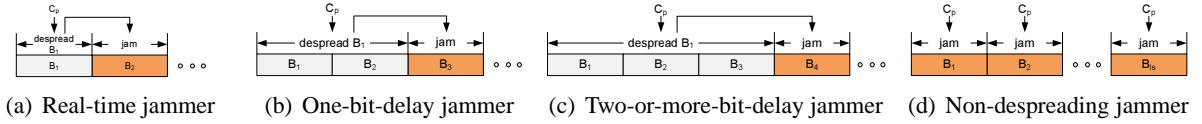


Figure 5: Reactive jamming with different capabilities

buffer. If the code with index sid_{l_s} can successfully despread the last l chips to get a bit value b_{l_s} , the sliding window potentially covers a seed.

The receiver despreads the seed bit-by-bit from the end to the beginning. After getting b_{l_s} , the receiver uses sid_{l_s} and b_{l_s} as the input to function F to get id_{l_s-1} , the index of the subset of C_p used for bit b_{l_s-1} . The receiver then sequentially tries each code in this subset to despread the l chips for bit b_{l_s-1} , until it finds the correct code. Assume the index of this code is sid_{l_s-1} and the decoded bit value is b_{l_s-1} . The sender then repeats this process to decode the earlier bits b_{l_s-2}, \dots, b_1 , and eventually reconstructs the seed $b_1||b_2||\dots||b_{l_s}$.

During this process, if any despreading failure occurs, the receiver gives up the current decoding process and shifts the sliding window by 1 chip to the right to look for the next seed candidate. Once the receiver gets the seed $b_1||b_2||\dots||b_{l_s}$, it uses this seed to generate the spreading code sequence for the message body and despreads the message body as discussed in Section 4.

5.4 Analysis

The objective of our analysis is to understand (1) the effectiveness of content-based code subset selection in enhancing DSD-DSSS's anti-jamming capability, and (2) the capability of this mechanism against DoS attacks discussed in Section 4.4.

5.4.1 Effectiveness against Jamming Attacks

We analyze the probability of an attacker jamming the seed to show the effectiveness of content-based code subset selection. Moreover, this scheme also increases the difficulty for a jammer to identify the right spreading code compared with a normal receiver. We thus analyze the search space (i.e., the set of candidate spreading codes) for both a receiver and a jammer to demonstrate the advantage of a normal receiver over a jammer.

We consider jammers with four levels of computation capabilities: (1) real-time, (2) one-bit-delay, (3) two-or-more-bit-delay, and (4) non-despreading jammers. All these jammers are reactive jammers that can synchronize with the sender. The first three types of jammers perform despreading and online analysis to assist jamming, which improves the jamming probability by reducing the number of candidate spreading codes (i.e., possible codes used by the sender).

As illustrated in Figure 5(a), a real-time jammer has intensive computation power to finish the analysis and identify the spreading code used for bit 1 (represented by chips B_1), and can use this information to jam the immediately following bit (represented by chips B_2). As shown in Figures 5(b) and 5(c), a one-bit-delay jammer and a two-or-more-bit-delay jammer need additional time, equivalent to the time for transmitting 1 bit and 2 or more bits, respectively, to finish online analysis before applying the result for jamming purposes. Thus, after learning the spreading code for bit 1, a one-bit-delay jammer and a two-or-more-bit-delay jammer can only jam bit 3 (represented by chips B_3) and bit 4 (represented by chips B_4) or later, respectively. These jammers may certainly perform the same analysis of every bit they receive and use the analysis result to jam future bits. A non-despreading jammer simply skips the despreading step and use C_e to jam the last bit of the seed and use C_p to jam the remaining part of the seed, as Figure 5(d) shows.

In the following, we prove Lemma 1 to assist the analysis.

Lemma 1. *Given k distinct subsets, the number of codes that can be used to derive these subsets by applying function F is in the range of $[k, \min\{2k, n^2 + n + 1\}]$.*

Proof. Since the output of function F is evenly distributed when the inputs are evenly distributed, for each subset, there are two possible codes as inputs. For each code, there are two possible subsets as outputs. Thus, the lower

bound is k and the upper bound is $\min\{2k, n^2 + n + 1\}$. □

Real-time Jammers: If a jammer can despread each bit in real-time (e.g., by using parallel computing devices), the jammer can know the code for despreading B_i once the transmission of B_i is complete. As Figure 6 shows, the jammer can then identify all $n+1$ subsets that contain this code. By using the inverse of function F , the jammer can also identify all possible codes in C_p that were used to determine these subsets, which were also used to spread b_{i+1} into B_{i+1} . The number of possible codes for B_{i+1} is in the range of $[n+1, 2(n+1)]$, according to Lemma 1. Thus, the jammer can jam the transmission of B_{i+1} by randomly selecting a code from these codes (rather than from C_p). Since the last bit of the seed is spread using codes in C_e , the number of all possible codes for the jammer is thus in the range of $[n+1, \min\{2(n+1), |C_e|\}]$.

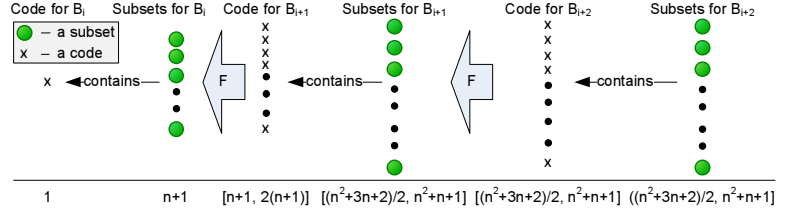


Figure 6: Jammers' view of spreading codes and subsets (Assume the jammer just derived code x for bit b_i (chips B_i))

In the worst case, a real-time jammer can despread all bits of the seed except for B_{l_s} and jams all bits. The jamming probability of the first bit is at most $\frac{1}{|C_p|}$, the jamming probability of the last bit is at most $P_{e0} = \frac{1}{n+1}$, and the jamming probability of B_i ($2 \leq i \leq l_s - 1$) is at most $P_{p0} = P_{e0} = \frac{1}{n+1}$. Thus, the jamming probability of the seed is at most

$$P_{\text{real-time}} = 1 - \left(1 - \frac{1}{|C_p|}\right) (1 - P_{p0})^{l_s-1}.$$

By including an ECC that can tolerate 1 bit error, we can reduce the maximum jamming probability to

$$P_{\text{real-time}} = 1 - (1 - P_{p0})^{l_s-1} - (l_s - 1) \left(1 - \frac{1}{|C_p|}\right) P_{p0} (1 - P_{p0})^{l_s-2}.$$

It is easy to see that the total search space for a real-time jammer throughout all bits of the seed is at least

$$SS_{\text{real-time}} = |C_p| + (l_s - 2)(n + 1) = n^2 + (l_s - 1)n + (l_s - 1).$$

Non-real-time Jammers: The results for one-bit-delay, two-or-more-bit-delay, and non-despreading jammers can be derived similarly. Due to the space limit, we do not show the details but list the final results for the jamming probabilities and search spaces in Table 1 and Table 2, respectively.

Table 1: Jamming probabilities for jammers with different jamming capabilities ($|C_p| = n^2 + n + 1$; $C_e = \left\lceil \frac{|C_p|}{2} \right\rceil$;

$$P_{p0} = \frac{1}{n+1}; P_{p1} = \frac{2}{(n+1)(n+2)}; P_{e1} = P_{e2} = \frac{1}{|C_e|}; P_{p2} > \frac{2}{(n+1)(n+2)}$$

real time	$n^2 + (l_s - 1)n + (l_s - 1)$
1 bit delay	$1 - \left(1 - \frac{1}{ C_p }\right)^2 (1 - P_{p1})^{l_s-3} (1 - P_{e1})$
q bits delay ($q \geq 2$)	$1 - \left(1 - \frac{1}{ C_p }\right)^{q+1} (1 - P_{p2})^{l_s-q-2} (1 - P_{e2})$
non-despreading	$1 - \left(1 - \frac{1}{ C_p }\right)^{l_s-1} \left(1 - \frac{1}{ C_e }\right)$
real time, tolerate 1 bit error	$1 - (1 - P_{p0})^{l_s-1} - (l_s - 1) \left(1 - \frac{1}{ C_p }\right) P_{p0} (1 - P_{p0})^{l_s-2}$
1 bit delay, tolerate 1 bit error	$1 - \left(1 - \frac{1}{ C_p }\right)^2 (1 - P_{p1})^{l_s-3} (1 - P_{e1}) - \frac{2}{ C_p } \left(1 - \frac{1}{ C_p }\right) (1 - P_{p1})^{l_s-3} (1 - P_{e1}) - (l_s - 3) \left(1 - \frac{1}{ C_p }\right)^2 P_{p1} (1 - P_{p1})^{l_s-4} (1 - P_{e1})$
q bits delay ($q \geq 2$), tolerate 1 bit error	$1 - \left(1 - \frac{1}{ C_p }\right)^{q+1} (1 - P_{p2})^{l_s-q-2} - (q+1) \frac{1}{ C_p } \left(1 - \frac{1}{ C_p }\right)^q (1 - P_{p2})^{l_s-q-2} (1 - P_{e2}) - (l_s - q - 2) \left(1 - \frac{1}{ C_p }\right)^{q+1} P_{p2} (1 - P_{p2})^{l_s-q-3} (1 - P_{e2})$
non-despreading, tolerate 1 bit error	$1 - \left(1 - \frac{1}{ C_p }\right)^{l_s-1} - (l_s - 1) \frac{1}{ C_p } \left(1 - \frac{1}{ C_p }\right)^{l_s-2} \left(1 - \frac{1}{ C_e }\right)$

Comparison of Jamming Probabilities:

Figure 7 shows the maximum jamming probabilities of the four types of jammers against the random seed with reasonable parameters. Recall that the size of C_p is determined by parameter n (i.e., $C_p = n^2 + n + 1$). Thus, we use parameter n as the x -axis in this figure. To better see the impact of ECC, we also include the maximum jamming probabilities assuming an ECC is used in the seed to tolerate 1 bit error in Figure 7.

Figure 7 shows that the real-time jammer has the highest jamming probability among all jammers. However, we would like to point out that the real-time jammer is a strong assumption; such a jammer may have to use special hardware (e.g., parallel computing devices) to obtain the despreading results. As the jammer has to tolerate 1 or 2 bit delays, the maximum jamming probability decreases significantly. Not surprisingly, the non-despreading jammer has the lowest jamming probability.

Figure 7 also shows that increasing n (and thus $|C_p|$) can quickly reduce the maximum jamming probability for all types of jammers. Moreover, the application of ECC can also reduce the jamming probability effectively, though it introduces additional computational and communication overheads. For example, with an ECC tolerating just 1 bit error, we can reduce the real-time jammer's maximum jamming probability from 0.31 to 0.05 when $n = 169$. Further increasing n or the number of bit errors the ECC can tolerate can quickly reduce the maximum jamming probability to a negligible level.

Comparison of Search Spaces: Now let us compare the numbers of candidate spreading codes that a normal receiver and a reactive jammer have to consider, respectively. Such numbers represent the computational costs they have to spend. Since a receiver buffers the complete seed before despreading it, it can despread the last bit of the seed first to learn sid_{l_s} , and then infer the indexes of subsets for previous bits of the seed. The size of total search space for a receiver is thus $(l_s - 1)(n + 1) + |C_e|$. To show the advantage of a receiver over a jammer, we compute function $Adv = \frac{SS_j}{SS_r}$ for real-time, one-bit-delay, two-or-more-bit-delay jammers, where SS_j and SS_r are the sizes of the total search space for the jammer and the receiver, respectively. The larger Adv is, the more advantage the receiver has over the jammer.

Figure 8 shows the advantage of a receiver over the jammers. (The non-despreading jammer is not included, since she does not despread at all.) All jammers have larger search space than the receiver, and the gap grows wider when n increases. The real-time jammer remains the most powerful jammer; it can reduce the search space for the next bit dramatically by despreading the current bit, and thus has the smallest search space among all jammers, which is close to the receiver's search space. Nevertheless, Figure 8 considers the lower bound of the jammers' search space. Moreover, there is still observable difference between the search spaces of the real-time jammer and the receiver. The search spaces of the one-bit-delay and two-or-more-bit-delay jammers have almost the same size, which are significantly larger than that of the receiver.

Table 2: Search spaces

real time	$1 - \left(1 - \frac{1}{ C_p }\right) \left(1 - \frac{1}{n+1}\right)^{l_s-1}$
1 bit delay	$2(n^2 + n + 1) + (l_s - 4) \frac{(n+1)(n+2)}{2}$
q bits delay ($q \geq 2$)	$(q + 1)(n^2 + n + 1) + (l_s - 2(q + 1)) \frac{(n+1)(n+2)}{2}$

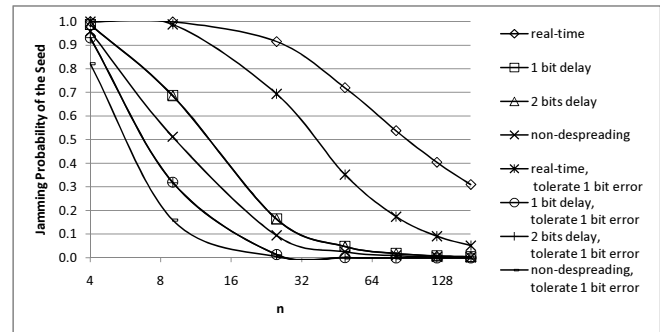


Figure 7: Maximum jamming probability against seed ($n = 4, 9, 25, 49, 81, 121, 169$; $l_s = 64$; $|C_e| = \left\lceil \frac{|C_p|}{2} \right\rceil$)

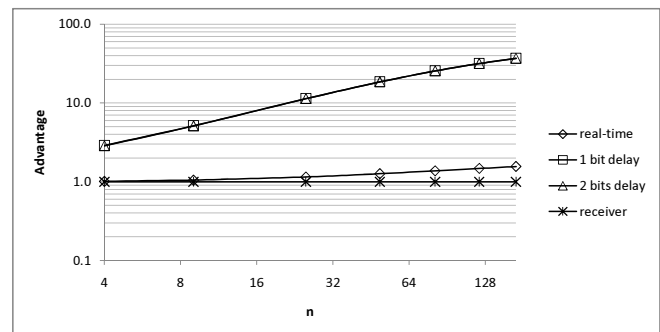


Figure 8: Advantage of receivers over jammers ($n = 4, 9, 25, 49, 81, 121, 169$; $l_s = 64$; $|C_e| = \left\lceil \frac{|C_p|}{2} \right\rceil$)

5.4.2 Effectiveness against DoS Attacks

As discussed in Section 4.4, a jammer can transmit bogus seeds or even entire bogus messages. As long as the communication channel is available to attackers, they can always inject bogus messages. Thus, in general, this is an unavoidable problem in presence of compromised receivers. When these bogus seeds are not concurrently transmitted and do not overlap with the sender's normal seed transmission, a receiver can filter them out using error detection coding and broadcast authentication (e.g., digital signature). However, when the bogus seeds do overlap with the normal seed, the receiver will have to consider all combinations of options for each bit of the seed, thus suffering from serious DoS attacks.

The proposed content-based code subset selection scheme can effectively mitigate such situations by chaining the codes used to spread different bits of the seed. To demonstrate the effectiveness of this approach, we show the number of candidate seeds when the jammer synchronizes with a sender and transmits a bogus seed ($B'_1||B'_2||\dots||B'_{l_s}$) to interfere with the transmission of the actual seed ($B_1||B_2||\dots||B_{l_s}$), as shown in Figure 9.

Intuition: During seed recovery, a receiver will attempt to recover the seed starting with both B_{l_s} and B'_{l_s} . The number of seed candidates is the number of paths starting from B_{l_s} or B'_{l_s} and ending at B_1 or B'_1 . In the basic DSD-DSSS, the receiver will try all possible paths shown in Figure 9. However, the content-based code subset selection scheme can constrain the paths between two seeds (dashed lines) during despreading. Intuitively, the jammer does not know which code subset is used to spread each bit of the seed at the time of its transmission, and thus cannot select the right code, which will be considered valid by a receiver during despreading. If the code for the i -th bit ($1 \leq i \leq l_s$) of the bogus seed is not in the subset for the i -th bit of the good seed, the receiver will not consider it for despreading the i -th bit of the bogus seed. As a result, the path from the good seed to the bogus one (in black dashed lines) will not exist. Similarly, if the code for i -bit of the good seed is not in the subset for i -th bit of the bogus seed, the receiver will not consider it for despreading the i -th bit of the good seed. Thus, the path from the bogus seed to the good one (in red dashed lines) will not exist.

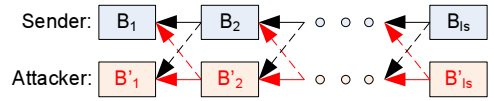


Figure 9: Seed recovery in presence of bogus seed transmission

During the analysis, we consider *non-despreading*, *real-time*, *one-or-more-bit-delay* jammers to see the best-case scenarios for the jammers when they can benefit from knowing a part of the seed and spreading codes. The capability of these jammers is the same as discussed earlier during the analysis of jamming probabilities. However, the objective of these jammers now is to trigger the receiver to have more seed candidates during despreading by injecting bogus seeds. We assume these jammers can perform despreading and transmitting operations at the same time, though they can only use the despreading results of each bit for later bits.

Non-despreading Jammers: If the jammer follows the sender's procedure to send the seed, the probability of having a path from from B'_{i+1} to B_i (red dashed line) and the probability of having a path from from B_{i+1} to B'_i (black dashed line) are both $\frac{1}{n^2+n+1}$, because any pair of codes only exist in exactly one subset. Only one among the $n^2 + n + 1$ subsets can despreading the i -th bit of both the bogus and the good seeds. The expected number of seed candidates is thus $2(1 + \frac{1}{|C_e|})(1 + \frac{1}{|C_p|})^{l_s-2}$ according to Theorem 4. The proof of Theorem 4 is omitted due to the space limit.

Theorem 4. *When there is a non-despreading jammer launching the DoS attack against seed disclosure, the expected number of seed candidates is $2(1 + p_1)(1 + p_2)^{l_s-2}$. Among them, $(1 + p_1)(1 + p_2)^{l_s-2}$ paths end at B_1 , and $(1 + p_1)(1 + p_2)^{l_s-2}$ paths end at B'_1 , where $p_1 = \frac{1}{|C_e|}$ and $p_2 = \frac{1}{|C_p|}$.*

Real-time and one-or-more-bit-delay Jammers: Similar to the analysis for non-despreading jammer, we analyze the expected number of seed candidates caused by real-time and one-or-more-bit-delay jammers. Due to the space limit, we simply list results and omit proofs. The expected number of seed candidates caused by real-time jammer is smaller than $2(1 + \frac{1}{|C_p|})(1 + \frac{n}{(n+1)^2})^{l_s-2}$, and that caused by one-or-more-bit-delay jammer is smaller than $(1 + 2p_2)(1 + p_2)E_3 + (p_4 + 2p_2)(1 + p_2)E'_3$, where

$$E_3 = 1 + \frac{p_4}{\lambda_1 - \lambda_2} \cdot \frac{(2p_5 - \lambda_2)(1 - \lambda_1^{l_s-3})}{1 - \lambda_1} + \frac{p_4}{\lambda_1 - \lambda_2} \cdot \frac{(\lambda_1 - 2p_5)(1 - \lambda_2^{l_s-3})}{1 - \lambda_2},$$

$$E'_3 = \frac{2p_5 - \lambda_2}{\lambda_1 - \lambda_2} \cdot \lambda_1^{l_s - 3} + \frac{\lambda_1 - 2p_5}{\lambda_1 - \lambda_2} \cdot \lambda_2^{l_s - 3}, p_2 = \frac{1}{|C_p|}, p_4 = \frac{2}{n+2}, p_5 = \frac{2n(n+3)}{(n+1)^2(n+2)^2}, \lambda_1, \lambda_2 = \frac{1+p_5 \pm \sqrt{(1+p_5)^2 - 4(1-p_4)p_5}}{2}.$$

Comparison: Figure 10 shows the expected numbers of seed candidates caused by non-despreading, real-time, and one-bit-delay jammers when they launch DoS attacks against seed disclosure. The more seed candidates the receiver has, the more computational cost the receiver has to spend receiving a message. Among three of them, the real-time jammer has the highest impact. However, it is still limited when n is reasonably large. The number of seed candidates is below 10 for all jammers when $n \geq 49$. The non-despreading jammer and the one-bit-delay jammers do not introduce much overhead to the receiver. The expected number of seed candidates by the non-despreading jammer is below 4 when $n \geq 9$. The expected number of seed candidates by the one-bit-delay jammer is below 1.5 when $n \geq 9$. When $n = 169$, the expected number of seed candidates of non-despreading, real-time, and one-bit-delay jammers are only 2, 2.87, and 1.01, respectively. Note that the lines shown in Figure 10 are conservative estimates showing the upper bound of the expected impact these jammers can introduce.

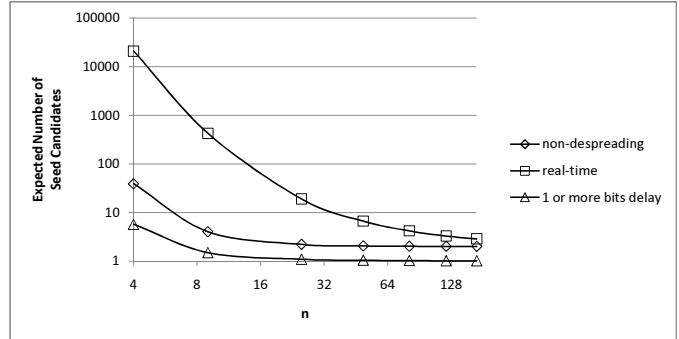


Figure 10: Expected number of seed candidates for normal receiver under DoS attacks against seed disclosure ($n = 4, 9, 25, 49, 81, 121, 169$; $l_s = 64$; $|C_e| = \left\lceil \frac{|C_p|}{2} \right\rceil$)

Compared with the basic DSD-DSSS scheme, in which the jammer can introduce 2^{l_s} seed candidates (e.g., 2^{64} seed candidates using the same parameters in Figure 10), the content-based code subset selection scheme has significantly reduced the impact of the DoS attacks against seed disclosure. Thus, it provides effective defense against such DoS attacks.

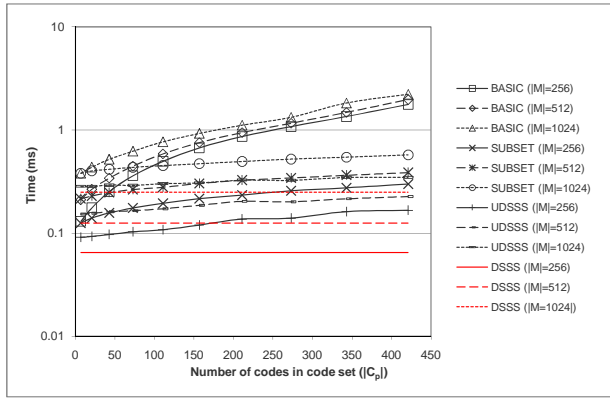
6 Experimental Evaluation

We have implemented a prototype of DSD-DSSS based on GNU Radio [1] using Universal Software Radio Peripherals (USRPs) with XCVR2450 daughter boards [11]. Our implementation includes both the basic DSD-DSSS scheme (named DSD-DSSS BASIC) and the enhanced DSD-DSSS with content-based code subset selection (named DSD-DSSS SUBSET). We have also implemented DSSS [6] and UDSSS [13] as references in our experimental evaluation.

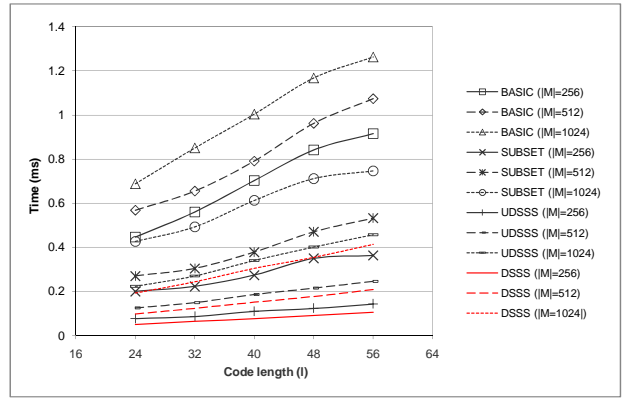
In our experiments, we used two USRPs with XCVR2450 daughter boards, one as the sender, and the other as the receiver. The sender was connected to a laptop (Intel Core 2 Duo @ 2.6GHz), while the receiver was connected to a desktop PC (Intel Pentium 4 @ 3.2GHz), both through 480 Mbps USB 2.0 links. Both the laptop and the desktop ran Ubuntu 9.04 and GnuRadio 3.2. The payload size in spreading/despreading module was configured to be 256, 512, or 1024 bits. We measured the receiver's average despreading time of a message for 200 rounds. Since messages were sent consecutively, the despreading of all messages after the first message was automatically synchronized (i.e., knowing the starting chip of each message). For DSD-DSSS, we set the seed size as 64 bits and used SAS v9.1.3 [15] to generate BIBD subsets of C_p . We used SHA-1 to as the pseudorandom number generator for both DSD-DSSS and UDSSS schemes.

Figure 11(a) shows the average despreading time of a message for DSD-DSSS BASIC, DSD-DSSS SUBSET, UDSSS, and DSSS schemes when using different size of code set. For DSD-DSSS, $|C_p| = n^2 + n + 1$, $|C_e| = \left\lceil \frac{|C_p|}{2} \right\rceil$, where $n \in [2, 20]$. For UDSSS, the number of code sequences is the same as the number of codes in $|C_p|$. As Figure 11(a) shows, DSSS is the most efficient scheme because only one code sequence is used to despread messages. UDSSS is slower than DSSS since it has to check the first code of all code sequences.

UDSSS is more efficient than DSD-DSSS because DSD-DSSS has to check $64 \cdot |C_e| = 64 \cdot \left\lceil \frac{|n^2 + n + 1|}{2} \right\rceil$ codes



(a) Time to despread message for different code set sizes ($l = 32$)



(b) Time to despread message for different code lengths ($|C_p| = 111$)

Figure 11: Comparison between DSSS, UDSSS, and DSD-DSSS

for BASIC scheme and $63 \cdot (n + 1) + |C_e| = 63 \cdot (n + 1) + \left\lceil \frac{n^2 + n + 1}{2} \right\rceil$ codes for SUBSET scheme, while UDSSS only needs to check $|C_p| = n^2 + n + 1$ codes. DSD-DSSS BASIC always has the largest number of codes to check. DSD-DSSS SUBSET scheme has larger number of codes to check than UDSSS when $n < 126$ (i.e., $|C_p| < 16003$). When $n \geq 126$, DSD-DSSS SUBSET scheme would be even more efficient than UDSSS. However, we cannot run the evaluation for $n \geq 126$ due to the large computational power requirement.

Figure 11(b) shows the average despreading time of a message for different code lengths ($l = 24, 32, 40, 48, 56$). It is obvious that all DSD-DSSS, UDSSS, and DSSS need more time to despread messages when the code length is increased. The despreading time of DSD-DSSS BASIC increases much faster than that of other schemes due to the much larger search space of codes. DSSS is still the most efficient scheme, and UDSSS is more efficient than DSD-DSSS. Although UDSSS is faster than DSD-DSSS in both Figure 11(a) and Figure 11(b), UDSSS suffers from the reactive jamming attack [13] while DSD-DSSS does not.

7 Related Work

Spread spectrum wireless communication techniques, including DSSS and FH, have been commonly used for anti-jamming communication [6]. However, as discussed earlier, traditional spread spectrum techniques all require pre-shared secret keys, and are not suitable for broadcast communication where there may be compromised or malicious receivers. We have discussed most closely related works in the introduction, including UFH and its variations [7, 16–18], UDSSS [13], and BBC [2, 3]. We do not repeat them here.

There are other related work, including approaches for detecting jamming attacks [21], identifying insider jammers [4, 5], mitigating jamming of control channels [9, 19], jamming avoidance and evasion [2, 20, 22], and mitigating jamming in sensor networks [10, 20]. Our technique is complementary to these techniques.

8 Conclusion

In this paper, we proposed DSD-DSSS, an efficient anti-jamming broadcast communication scheme; it achieves anti-jamming capability through randomly generating the spreading code sequence for a broadcast message through a random seed and delaying the disclosure of the seed at the end of the message. We also developed an effective protection for the disclosure of the random seed through content-based code subset selection. Our analysis in this paper demonstrated that this suite of techniques can effectively defeat jamming attacks in wireless broadcast communication. Our implementation and evaluation shows the feasibility of DSD-DSSS in real world.

In our future work, we will explore optimization techniques for DSD-DSSS and further evaluate its properties through experiments.

References

- [1] GNU Radio - The GNU Software Radio. <http://www.gnu.org/software/gnuradio/>.
- [2] L. Baird, W. Bahn, and M. Collins. Jam-resistant communication without shared secrets through the use of concurrent codes. Technical report, US Air Force Academy, 2007.
- [3] L. C. Baird, W. L. Bahn, M. D. Collins, M. C. Carlisle, and S. C. Butler. Keyless jam resistance. In *Proceedings of the IEEE Information Assurance and Security Workshop*, pages 143–150, June 2007.
- [4] J. Chiang and Y. Hu. Extended abstract: Cross-layer jamming detection and mitigation in wireless broadcast networks. In *Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '07)*, 2007.
- [5] J. Chiang and Y. Hu. Dynamic jamming mitigation for wireless broadcast networks. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, 2008.
- [6] A. Goldsmith. *Wireless Communications*. Cambridge University Press, August 2005.
- [7] T. Jin, G. Noubir, and B. Thapa. Zero pre-shared secret key establishment in the presence of jammers. In *Proceedings of MobiHoc '09*, May 2009.
- [8] D. L. Kreher and D. Stinson. *Combinatorial Algorithms: Generation, Enumeration, and Search*. CRC Press, 1999.
- [9] L. Lazos, S. Liu, and M. Krunz. Mitigating control-channel jamming attacks in multi-channel ad hoc networks. In *Proceedings of 2nd ACM Conference on Wireless Networking Security (WiSec '09)*, March 2009.
- [10] M. Li, I. Koutsopoulos, and R. Poovendran. Optimal jamming attacks and network defense policies in wireless sensor networks. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, 2007.
- [11] Ettus Research LLC. The USRP product family products and daughter boards. <http://www.ettus.com/products>. Accessed in October 2009.
- [12] R. Poisel. *Modern Communications Jamming Principles and Techniques*. Artech House Publishers, 2006.
- [13] C. Pöpper, M. Strasser, and S. Čapkun. Jamming-resistant broadcast communication without shared keys. In *Proceedings of the USENIX Security Symposium*, 2009.
- [14] J. Proakis. *Digital Communications*. McGraw-Hill, August 2000.
- [15] SAS. Business analytics and business intelligence software. <http://www.sas.com>.
- [16] D. Slater, P. Tague, R. Poovendran, and B. Matt. A coding-theoretic approach for efficient message verification over insecure channels. In *Proceedings of the 2nd ACM Conference on Wireless Networking Security (WiSec '09)*, pages 151–160, March 2009.
- [17] M. Strasser, C. Pöpper, S. Čapkun, and M. Čagalj. Jamming-resistant key establishment using uncoordinated frequency hopping. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pages 64–78, 2008.
- [18] M. Strasser, C. Pöpper, and S. Čapkun. Efficient uncoordinated FHSS anti-jamming communication. In *Proceedings of MobiHoc '09*, May 2009.
- [19] P. Tague, M. Li, and R. Poovendran. Probabilistic mitigation of control channel jamming via random key distribution. In *Proceedings of IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '07)*, pages 1–5, 2007.
- [20] W. Xu, W. Trappe, and Y. Zhang. Channel surfing: Defending wireless sensor networks from jamming and interference. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN '07)*, 2007.
- [21] W. Xu, W. Trappe, Y. Zhang, and T. Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '05)*, 2005.
- [22] W. Xu, T. Wood, W. Trappe, and Y. Zhang. Channel surfing and spatial retreats: Defenses against wireless denial of service. In *Proceedings of the 3rd ACM workshop on Wireless security (WiSe '04)*, 2004.