

# Zero Pre-shared Secret Key Establishment in the Presence of Jammers

Tao Jin  
College of Computer Science  
Northeastern University  
Boston, MA 02115  
taojin@ccs.neu.edu

Guevara Noubir  
College of Computer Science  
Northeastern University  
Boston, MA 02115  
noubir@ccs.neu.edu

Bishal Thapa  
College of Computer Science  
Northeastern University  
Boston, MA 02115  
bthapa@ccs.neu.edu

## ABSTRACT

We consider the problem of key establishment over a wireless radio channel in the presence of a communication jammer, initially introduced in [14]. The communicating nodes are not assumed to pre-share any secret. The established key can later be used by a conventional spread-spectrum communication system. We introduce new communication concepts called *intractable forward-decoding* and *efficient backward-decoding*. Decoding under our mechanism requires at most twice the computation cost of the conventional SS decoding and one packet worth of signal storage. We introduce techniques that apply a *key schedule* to packet spreading and develop a provably optimal key schedule to minimize the bit-despreading cost. We also use efficient FFT-based algorithms for packet detection. We evaluate our techniques and show that they are efficient both in terms of resiliency against jammers and computation. Finally, our technique has additional features such as the inability to detect packet transmission until the last few bits are being transmitted, and transmissions being destination-specific. To the best of our knowledge, this is the first solution that is optimal in terms of communication energy cost with very little storage and computation overhead.

## Categories and Subject Descriptors

C.2.1 [Computer-Communication Networks]: Design—*Wireless Communication*

## General Terms

Algorithms, Design, Security

## Keywords

Anti-jamming, Spread Spectrum, Zero Pre-shared Secret

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*MobiHoc'09*, May 18–21, 2009, New Orleans, Louisiana, USA.  
Copyright 2009 ACM 978-1-60558-531-4/09/05 ...\$5.00.

## 1. INTRODUCTION

Radio-Frequency wireless communication occurs through the propagation of electro-magnetic waves over a broadcast medium. Such broadcast medium is not only shared between the communicating nodes but is also exposed to adversaries. The resiliency against malicious behavior is obviously of significant importance for military communication in a battle-field. It is also rapidly gaining significance in civilian and commercial applications due to the increased reliance on wireless networks for connectivity to the cyber-infrastructure, and applications that monitor our physical infrastructure such as tunnels, bridges, landmarks, buildings, etc.

Jamming and anti-jamming techniques for the physical layer of wireless systems supporting mostly voice communication have been extensively studied for several decades [13]. However, it is only recently that the popularity of multi-hop data networks with complex medium sharing, coding, and application protocols opened the door for sophisticated attacks and resulted in the exploration of new resilience mechanisms. Emerging attacks include ultra low-power cross-layer attacks that aim at disturbing the operation of networks by targeting control-mechanisms such as packet routing, communication beacons or pilots, carrier sensing mechanism, collision avoidance exponential back-off mechanism, network topology, and size of the congestion control window. For example, by transmitting a few pulses at the right frequency, right time and right location, highly efficient (energy/computation wise) attacks can be deployed with off-the-shelf hardware [10, 2, 11, 4, 16, 5].

### 1.1 Motivation

Spread Spectrum (SS) is one of the most efficient mechanisms used for anti-jamming communication. Military systems, in particular, rely on SS systems along with antenna nulling, channel coding to counteract malicious attacks. In civilian systems however, SS has been discarded from usage mainly because it requires a pre-shared secret, which may not be available [1, 7]. For example, in systems which employ large number of dynamically associating/disassociating nodes, the pre-sharing has to be done over an open channel making it an easy target by an adversary who focuses all its jamming energy on the key establishment protocol. The problem has been introduced previously as the anti-jamming/key establishment circular dependency problem [14]. Strasser et al. also propose a new mechanism called Uncoordinated Frequency Hopping (UFH) to break this circular dependency, however, at a high communication cost.

In this paper, we propose a novel approach for breaking the anti-jamming/key establishment circular dependency with significant energy efficiency advantages over UFH. Our mechanism relies on two main properties: (1) intractable forward-decoding (preventing an adversary from detecting or decoding an on-going communication), (2) efficient backward-decoding (allowing any receiver to decode the time-reversed signals). Note that although the adversary can also decode the time-reversed signal (and find out which Pseudo random (PN) spreading sequence was used), it will be too late for it to jam by the time it retrieves the PN-sequence (See Figure 1). The basic idea behind our scheme is that the sender spreads the message with a cryptographically-strong PN-sequence. Forward-decoding the packet requires guessing the whole key initially, which we will show to be infeasible for the jammer to accomplish (by brute force) in time to jam the packet before the end of the packet transmission. As communication progresses, the entropy of the spreading sequence decreases (See Figure 4), thus on the receiver side, decoding the time-reversed version of the packet only requires the receiver to guess one bit of the key at each stage of the decoding process. We will also show that under our scheme, at each instant the time it takes for a node to brute-force the PN sequence plus the TX/RX turn-around time and propagation time is larger than the time it takes for the sender to send the remaining bits of the message. This makes forward-decoding intractable.

The main advantage of our solution, in comparison with UFH [14], is that it does not require extra energy for transmitting packets. It is in fact as energy efficient as the conventional SS communication where the communicating nodes pre-share a secret key. UFH, on the other hand, requires on average  $n$  times more energy than traditional SS,  $n$  being the spreading factor that is in the order of hundreds. We achieve this communication-energy efficiency with a *slight* increase in the receiver computation and storage cost. We show that the computation/decoding cost is at most twice the computation cost of conventional SS (See Theorem 2) and the storage required is of one packet length. A secondary advantage of our technique is the delayed communication detection, which makes it practically impossible for an adversary to sense an ongoing communication until it is “almost” over. This stealthiness further increases the inefficiency of the adversary by forcing it to be a channel-oblivious jammer [2].

## 1.2 Related Work

Anti-jamming techniques have been studied for decades [13]. Most of the earlier mechanisms however, only focussed on a physical layer protection and made use of SS techniques, directional antennas, and coding schemes. At the time, most wireless communication was not packetized nor networked. Furthermore, the small size of the networks then (mostly military), and the way they were deployed allowed for pre-configuration with shared secret keys to be possible.

Reliable communication in the presence of adversaries regained significant interest in the last few years. New attacks emerged with the advent of more complex applications and deployment environments. Several specifically crafted attacks and counter-attacks were proposed for: packetized wireless data networks [11, 10], multiple access resolution in the presence of adversaries [3, 2, 1], multi-hop networks [16, 15, 10], broadcast communication [6, 5], cross-layer attacks [11],

and navigation information broadcast [12]. While many recently proposed countermeasure techniques can (and are assumed to) be layered on a SS physical layer, it is usually taken for granted that the communicating nodes pre-share a secret key. Strasser et al. recognized this as a significant impediment to the use of SS, even when the communicating nodes possess public keys and certificates that potentially allow them to setup a shared secret key [14].

Strasser et al. proposed UFH, a technique for establishing a symmetric secret key in the presence of adversaries. In UFH, the sending node hops at a relatively fast rate (e.g., 1600 hops per second) over  $n$  channels. It repeatedly sends fragments of the mutual authentication and key establishment protocol. The receiver hops, on the other hand, are significantly slower. Therefore, although the receiver does not know the sender’s hopping sequence, statistically, it can receive  $1/n$  of the sent packets. The authors show that an adversary has a very low probability of jamming these packets. They build upon this basic mechanism to construct a jamming-resilient mutual authentication and key establishment protocol. Their paper introduces the first reliable key establishment protocol for SS without a pre-shared secret. However, unlike traditional SS systems with pre-shared keys, the proposed mechanism incurs an increase in energy cost by a factor of  $n$  due to the implicit redundancy in packet transmissions (retransmissions of message fragments that are not received) required by their scheme. This is the closest work related to our paper. Our mechanism, unlike UFH, retains the main benefits of the original SS communication in terms of communication energy (all transmitted energy is used in the packet decoding process). It does incur a higher computation cost, which we show later is no more than twice the cost of the traditional SS with pre-shared secret. With ever increasing computation power of computers today, this is a negligible issue.

Other countermeasure techniques discard the possibility of using SS because of the narrow RF bands available to ad hoc networks, or because of the absence of a pre-shared key as mentioned above [7, 1]. These techniques are much less energy efficient than SS. Note that SS can still be used in narrow band if the signal is spread in time at no additional energy cost. The tradeoff in that will be a reduced data rate by a factor equal to the spreading length, which is not necessarily a limitation as two nodes can have multiple simultaneous communications as in Code Division Multiple Access systems.

## 1.3 Contributions

The contributions of this paper are both conceptual and algorithmic:

- Zero communication-energy overhead key establishment of a shared key without pre-agreed knowledge (in comparison with conventional SS with pre-shared keys): a novel approach based on intractable forward-decoding and efficient backward-decoding.
- Undetectable communication until end of transmission (delayed detection) forcing the jammer to become energy-inefficient and channel-oblivious [2].
- A destination-oriented scheme that prevents efficient simultaneous-attacks on multiple receivers.

- Computationally efficient end of the message detection (a FFT-based technique), and message extraction (use of a key-scheduling algorithm that requires at most twice the decoding cost of conventional SS).

## 2. SETUP MODEL

Our setup model considers systems that are traditionally capable of doing SS, such as mobile ad hoc network. In particular, it is not applicable to systems with low computational power, e.g., wireless sensor network. The implementation of our scheme requires systems to have at least a GB of memory to carry out FFT computations.

### 2.1 System Model

We consider a wireless communication network where several nodes are trying to establish pairwise-shared secret that would enable SS communication. Our model and the problem formulation is very similar to [14]. We focus on a pair of communicating nodes along with a jammer, all sharing a RF channel. The jammer’s objective is to prevent the establishment of the secret key between the communicating nodes, because once this key is established, the communicating nodes can use conventional SS for communication making them resilient to jamming. Our main objective is to devise a jammer-resilient message-delivery mechanism with no pre-shared information, which can be used by any Mutual Authentication and Key Agreement Protocol (MAKAP) to deliver few messages and establish a key for future SS communication. In this paper, we consider the same MAKAP as in [14], namely Elliptic Curve Diffie Hellman (ECDH) because of the small number of messages exchanged (two) and their short length. Our method uses Direct-Sequence SS (DSSS), but it easily generalizes to Frequency-Hopping SS (FHSS).

#### Assumption

- We assume that there exists a trusted Certificate Authority (CA) that issues digital certificates attesting each user’s public key.<sup>1</sup>
- Anything that is known to the receiver about the protocol and the sender is known to the jammer. This includes the encoding the decoding mechanism that makes our system so efficient.<sup>2</sup>

### 2.2 Adversary Model

We consider an adversary that is co-located with the sender and the receiver that can jam, replay previously collected messages, insert fake messages and/or modify bits of the message. The primary goal of the adversary is to prevent successful reception of the sender’s message by the receiver. However, in an attempt to do so, a jammer may simply increase the delay of the message extraction process or cause denial of service (DoS) attack on the receiver side. So, it’s secondary goal may very well be to increase the computation and energy cost of the receiver while minimizing its

<sup>1</sup>Note that given the energy, computation, and storage efficiency of our techniques, if no certification authority is available, we can consider using our scheme to transmit all packet without ever establishing a key.

<sup>2</sup>Regardless of the attacker being one of the participating nodes or an outside jammer, it has the same amount of information available to the receiver.

own jamming cost. We define jammer’s performance as the trade-off function relating the packet loss rate (PLR) with the total jamming cost. Our classification of the adversary attacks is inspired by the well known *active* attack categorization and the attacker model of [14]. However, the specific attacker strategies we designed and implemented for evaluation of our scheme are protocol-specific. In Section 5, we also present the empirical optimal jammer strategy and show that it is cost-inefficient under our proposed scheme.

#### Assumption

- We ignore the gain of configuring physical layer parameters such as antenna gains, coding schemes, and power-control (e.g., near-far problem) since they can be optimized the same way as they are optimized in conventional SS, independent of our mechanism.
- Our model does not consider the case where the jammer can block the propagation of the radio signal (e.g., by putting a node into a Faraday’s cage)
- We assume that the adversary cannot tunnel the channel signals for remote brute-forcing before the end of the packet transmission (few milliseconds).

#### Taxonomy of the Attacks

1. **Jamming:** The attacker can jam the communication link in various ways, such as sending a high-power pulse either at periodic intervals, continuously, or in a memoryless fashion [2]. The goal is to distort packets and cause failure of correct packet decoding.
2. **Replay Attack:** The attacker can replay previously captured communication messages. The goal is to increase the computation cost of (1) packet decoding, and/or (2) signature verification.
3. **Targeted Modification:** The attacker can modify some bits of the message by focusing the jamming energy on some portion of the message. The jammer cannot deterministically carry out this attack since it can not detect on-going communication under our mechanism until last few bits of message are sent.
4. **Computation Denial of Service:** The attacker inserts partial or complete messages to overwhelm receiver’s (1) packet decoding, and/or (2) signature verification.

Notice that we do not make any additional assumption on the limitation of jammer’s computation power and energy more than what a traditional SS does. Obviously, if the jammer is infinitely powered energy-wise and continuously jams all the time, it could reduce the throughput to 0%, just like it would in a traditional SS. Our main goal here is to devise a jammer resilient key establishment protocol with no pre-shared key and at no additional cost compared to the traditional SS.

In Section 5, we evaluate the performance of the jammer types described above and present simulation results.

### 3. TREKS IN DSSS

Time-Reversed Message Extraction and Key Scheduling (TREKS) is a communication approach based on zero pre-shared key spread spectrum (ZPKS), specifically DSSS (ZPK-DSSS) in this paper. We will first present the core idea of ZPK-DSSS and its efficiency against jamming. Then we propose a novel key scheduling scheme, which enables efficient backward-decoding making TREKS very applicable to systems in terms of communication energy, computation and storage cost.

#### 3.1 Zero pre-shared key DSSS

Assume that sender S, receiver R, and jammer J all share the same channel. Let  $M$  denote the message that S wants to send to R,  $l$  the bit-length of  $M$ . Prior to the start of transmission, S randomly generates a secret key  $K$  of  $k$  bit length. Unlike conventional DSSS,  $K$  is not known to anyone but S when communication occurs. S generates a cryptographically strong PN-sequence using  $S$  and spreads  $M$ . Although, a PN-sequence cryptographically generated from the key (as a seed to a symmetric encryption algorithm such as AES) are not optimal in terms of orthogonality, they perform reasonably well and have been used in many military SS systems [13].

In conventional DSSS, R keeps attempting to despread incoming signals with the key, that is pre-shared between S and R, until it detects the beginning of the message, then the forward-decoding of whole message starts. In ZPK-DSSS, there is no pre-sharing of the key. Thus, R needs to first identify the key  $K$  chosen by S. Without knowing  $K$ , R does not even know when the DSSS communication occurred. The only possibility is to brute force all possible keys on every chip of the incoming signal until a key is found that could properly decode the complete message. Given that the key size is  $k$  bit, the complexity of exploring the key space by brute force is  $O(2^k)$ . This cost is infeasible for real-time communication. In Section 3.3, we introduce our backward-decoding mechanism with a key schedule to be integrated into this approach that make its efficient and viable for real-time communication.

#### 3.2 Jamming resiliency

We first demonstrate the fundamental strengths of the proposed approach from the standpoint of key recovery intractability and energy efficiency against jammers.

##### 3.2.1 Communication energy efficiency

In this section, we present the way the message bits are spread and how the total energy per packet is preserved. We also show that the energy cost of the jammer to counter the effect of spreading increases by factor of  $n$ .

Symbol	Definition
$d \in \{+1, -1\}$	BPSK symbols that are estimated and mapped to $\{0,1\}$ equiprobably
$\hat{d} \in \{+1, -1\}$	Received BPSK symbols that are estimated and mapped to $\{0,1\}$
$n$	Spreading Factor
$pn_i, 2n_i, \dots, ng \in \{-1, +1\}$	$i^{th}$ chip of cryptographically designed SSEQ unknown to adversary.
$E_b$	Energy per transmitted bit assuming w.l.o.g 1 bit sent per unit time.
$u_i = d\sqrt{\frac{E_b}{n}}pn_i$	BPSK modulated signal transmitted by the sender.
$J$	Jammer energy per unit time
$I_j, 2n_i, \dots, ng$	Adversary's transmitted signals indexed at the chip level.
$\frac{J}{n}$	Mean square of $I_j$
$v_i$	Received signal indexed at chip level.
$r_i$	jamming chip with unit mean square.
$BER(x, y, z)$	Bit Error Rate of despread signal when $E_b = x$ , $J = y$ and $n = z$

Table 1: Terminology

**FACT 1.** Spreading a signal by a factor  $n \gg 1$  allows, the communicating nodes to counter an  $n$ -times stronger jammer at no extra-energy cost for the sender:

**Proof.** Since, we are only interested in the impact of jamming, we normalize the path loss and antenna gains to 1. For simplicity, we ignore thermal (white) noise. The same result still holds in the general case. By definition,

$$v_i = u_i + I_i = d\sqrt{\frac{E_b}{n}}pn_i + \sqrt{\frac{J}{n}}r_i$$

Consider the following decoding technique<sup>3</sup>:

$$\hat{d} = 1 \text{ iff } \sum_{i=1}^n v_i pn_i > 0$$

We consider BPSK modulation but the results generalize to other modulations as well. Then  $BER(E_b, J, n)$

$$\begin{aligned} &= Pr[\hat{d} = 1 \text{ and } d = -1] + Pr[\hat{d} = -1 \text{ and } d = 1] \\ &= 2 * Pr[\sum_{i=1}^n v_i pn_i > 0 \text{ and } d = -1] \\ &= 2 * Pr[d\sqrt{\frac{E_b}{n}} \sum_{i=1}^n pn_i pn_i + \sqrt{\frac{J}{n}} \sum_{i=1}^n r_i pn_i > 0 \text{ and } d = -1] \\ &= 2 * Pr[-\sqrt{\frac{E_b}{n}} \sum_{i=1}^n pn_i pn_i + \sqrt{\frac{J}{n}} \sum_{i=1}^n r_i pn_i > 0 \text{ and } d = -1] \\ &= 2 * Pr[-\sqrt{E_b n} + \sqrt{\frac{J}{n}} \sum_{i=1}^n r_i pn_i > 0] * Pr[d = -1] \\ &= Pr[\sum_{i=1}^n r_i pn_i > \sqrt{\frac{E_b}{J} n}] \end{aligned}$$

where  $pn_i$  is a random variable independent from the adversary's  $r_i$  choices. Therefore,  $\sum_{i=1}^n r_i pn_i$  is the sum of  $n$  random variables of equal probability taking values  $\{-1, +1\}$ . The distribution of the sum can be derived from the Binomial distribution. For  $n \gg 1$ , this distribution can be approximated by a Normal distribution of zero mean and variance  $n$ :  $N(0, n)$ . Thus,

$$\begin{aligned} BER(E_b, J, n) &= \int_{\sqrt{\frac{E_b}{J} n}}^1 \frac{1}{\sqrt{2\pi n}} e^{-\frac{x^2}{2n}} dx \\ &= \int_{\sqrt{\frac{E_b}{J} n}}^1 \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx \end{aligned} \quad (1)$$

Eq. (1) indicates that when the spreading factor is increased by a factor  $c$ , the adversary needs to scale its jamming energy  $J$  by a factor  $c$  to maintain the same  $BER$ . On the transmitter side, since the energy per bit is kept constant, transmitter still spends the same amount of energy while being resilient to  $c$  times more jamming.  $\square$

##### 3.2.2 Computational infeasibility for jammer

In order to jam in a cost efficient way, the jammer needs to identify the spreading key. As shown above, the complexity of finding the key is  $O(2^k)$ . If  $k$  is designed such that identifying the key takes significantly more time than the packet transmission then even if the jammer eventually finds the key, it is too late to jam the packet as the transmission is already over. For example, given a key size of

<sup>3</sup>Note that we are assuming that the receiver knows the bit synchronization. This is a common assumption in analyzing SS systems. We will see in Section 4 how this is achieved.

$k = 20$ ,  $n = 100$ , and sender chip rate of 100Mcps (10ns chip duration), it takes few milliseconds to transmit (e.g., 1 ms for 1000 bits spread with  $n = 100$ ). That means it requires the jammer in the order of 10 multiplication operations per picosecond to brute-force  $2^{20}$  possible keys within few milliseconds of transmission time, which is not possible for a field deployed jammer to accomplish. We call this *intractable forward-decoding*, which is illustrated in Figure 1.

### 3.2.3 Limitations

Intractable forward-decoding is due to zero pre-shared secret in ZPK-DSSS, which in-turn also applies to the receiver. Since the receiver needs to try  $O(2^k)$  possibilities on each incoming chip signal to figure out the spreading key, it causes a considerably high computation overhead. This is a major limitation of the basic ZPK-DSSS.

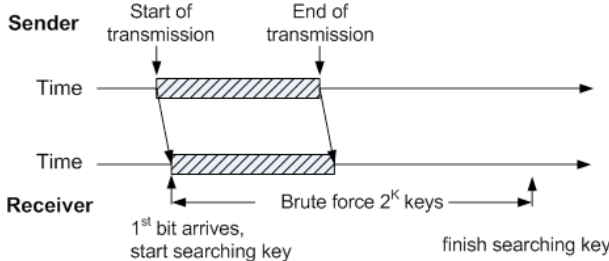
In the following section, we introduce a novel spreading key scheduling scheme, which builds upon ZPKS and enables both *intractable forward-decoding* and *efficient backward-decoding*. This drastically reduces the computation overhead for the receiver from  $O(2^k)$  to  $O(2k)$  while the jamming resiliency remains the same.

## 3.3 Key scheduled reverse-time decoding

### 3.3.1 Key size vs. jamming resiliency

Before delving into the details of our key scheduling scheme, we first show how the key-entropy is reduced as the transmission gets closer to the end, which significantly cuts down the cost to identify the key for the receiver but still requires the same effort from an adversary.

**Theorem 1.** *Let  $T_{trans}(l)$  denote the transmission time of  $l$  bits,  $T_s(k)$  the time required to brute force all possible  $k$  bit keys. Given a message  $M$  and key size  $k$ , if it is secure<sup>4</sup> to spread  $M$  with a  $k$ -bit key, it is secure to spread the last  $\frac{iM_j}{2^i}$  bits with  $k - i$  bit key, where  $i \leq \log_2(|M|)$ .*



**Figure 1: Message delivered before the key is broken.**

**Proof.** We first show that it is secure to spread the second half of  $M$  with  $k - 1$  bit key. Since it is secure to spread  $M$  with a  $k$  bit key, we have

$$\begin{aligned} T_{trans}(|M|) &\ll T_s(k) \\ T_{trans}\left(\frac{iM_j}{2^i}\right) &= \frac{1}{2}T_{trans}(|M|) \\ &\ll \frac{1}{2}T_s(k) = T_s(k-1) \end{aligned} \quad (2)$$

<sup>4</sup>In the rest of the paper, *secure key size* implies it takes significantly more time to brute-force all possible keys used to spread than to transmit the message.

Eq. (2) shows that it is secure to encode  $\frac{iM_j}{2^i}$  bits with  $k - 1$  bit key. Therefore, even if we use a 1-bit weaker key to encode the second-half of  $M$ , we can guarantee that the whole message can still be delivered before the jammer brute forces all possible keys. By induction, it is easy to get that  $T_{trans}\left(\frac{iM_j}{2^i}\right) \ll T_s(k - i)$ . Thus, it is secure to spread the last  $\frac{iM_j}{2^i}$  bits with  $k - i$  bit key.  $\square$

The intuition behind Theorem 1 is that as transmission goes on, less time is left for jammer to find the key, so it is safe to encode the rest of the message with slightly weaker keys<sup>5</sup>.

### 3.3.2 Spread key scheduling

Based on Theorem 1, we introduce a key scheduling scheme to TREKS. As shown in figure 2, instead of spreading the complete message with a fixed key, we partition the message into  $k$  segments (note that the segments are transmitted in a continuous way), where  $k$  is the key size. We call each segment “*schedule*”. The size of  $i$ th segment  $M_i$  is  $\lceil \frac{iM_j}{2^i} \rceil$ . At the start of spreading process, we use full length key to spread  $M_1$ . After each schedule, we set the most significant bit of the key to a known value and resume encoding the next segment with this 1-bit weaker key. We repeat this process until the last schedule, which is encoded with a key with only 1 bit secret. So, it is easy to see that the message length  $l$  has to be at least  $2^k$  such that  $k$  could be decreased to 1 bit towards the end of the key schedule. For simplicity, we assume that  $l = 2^k$ . We loosen this assumption in the later section. Algorithm 1 outlines the message segmentation and key scheduling.

Symbol	Definition
$M$	message to be transferred
$K$	secret key
$l$	length of message in bits
$k$	size of secret key in bits
$K[m \dots n]$	part of the $K$ from $m^{th}$ bit to $n^{th}$ bit
$M[m \dots n]$	part of the $M$ from $m^{th}$ bit to $n^{th}$ bit
$K_i$	key used in schedule $i$
$M_i$	message segment belonging to schedule $i$
$N_i$	size of rest of message at the start of schedule $i$

**Table 2: Summary of the notations.**

**Algorithm 1: Sender encoding message with key schedule.**

```

1.  $N_1 \leftarrow M$ 
2. for  $i = 1 \dots k$  do
    $K_i \leftarrow K[i \dots k]$ 
    $M_i \leftarrow N_i[1 \dots \lceil \frac{iM_j}{2^i} \rceil]$ 
   cryptographically generate  $PN_i$  from  $K_i$ 
   encode  $M_i$  with  $PN_i$ 
    $N_{i+1} \leftarrow N_i[|M_i| + 1 \dots |N_i|]$ 

```

**Intractable forward-decoding:** By the definition of the key schedule and theorem 1, the property of intractable forward-decoding is maintained.

**Efficient backward-decoding:** Due to the decreasing key entropy, it becomes easier for the receiver to identify the key as the transmission is closer to the end. Specifically, since the last key schedule has entropy of 1 bit, the receiver needs

<sup>5</sup>Additional measures can be taken to prevent overlap between weakened key spaces.

to try just two keys on each incoming chip to detect the potential end of message (EoM). Once the receiver detects a potential EoM, it starts inferring the key from previously received signals using the knowledge that the entropy of  $K_{j+1}$  from that of  $K_j$  in the key schedule increase by only a bit in reverse time. So the receiver needs to try  $2k$  keys in total before realizing all the  $k$  bits of the key, significantly lowering the cost of finding the key from the basic scheme.

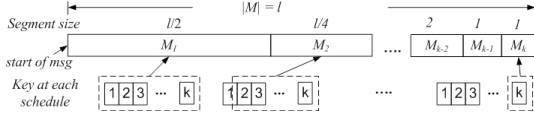


Figure 2: TREKS with key scheduling.

### 3.4 Further improvements and discussion

#### 3.4.1 MAC-masked key scheduling

In the key scheduling scheme above, the last scheduled key  $K_k$  is always either 0 or 1 for any sender/receiver pair. Thus, the jammer could jam with a PN-sequence generated by 0 or 1, which is likely to compromise the last message segment. Once the EoM is jammed and the receiver is not able to detect it, the reverse decoding cannot start. In order to tackle this issue, we take the receiver's MAC address to mask the key at each schedule. The revised key scheduling strategy is illustrated in figure 3. The key  $K_i$  used to encode  $M_i$  is generated by replacing the most significant  $i-1$  bits of the receiver's MAC address with the most significant  $i-1$  bits of  $K$ . It is easy to see that the hardness of the key inferring remains the same. Whereas, the last scheduled key is different across different receivers. Thus, the jammer can only target one receiver at a time. The potential jamming attack mentioned earlier becomes a destination-oriented attack.

In section 5 we will discuss the impact of the MAC jammer and show that our system is highly resilient against such jammer as it is against other jammers.

#### 3.4.2 Key scheduling with linear tail

As mentioned at the end of 3.3.2, we assumed that  $l = 2^k$  so that key size can be decreased down to 1 bit by  $k^{th}$  schedule, and total message length  $l$  for  $k = 20$  would be 1M bits. Obviously this is too large for a message size.

We also observed that if  $T_{trans}(|M|) \leq T_{\pm}$ , where  $T_{\pm}$  is the radio turn around time of the jammer, it is impossible for the jammer to jam  $M$ . In this case, when the jammer detects the transmission and switches to a transmit mode, the message has already been delivered. Take 802.11 as an example, the radio turn around time is  $10\mu s$ . Consider a

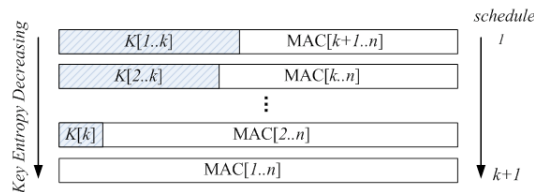


Figure 3: MAC-masked key scheduling.

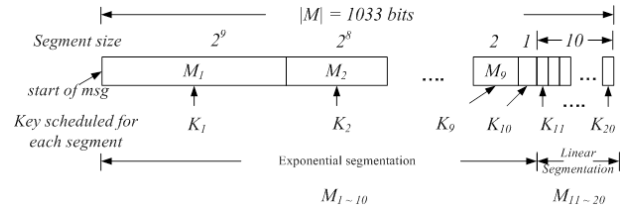


Figure 4: Key scheduling with linear tail.

spreading factor  $n = 100$ , chip rate of 100Mcps, then we have  $T_{trans}(1) = 1\mu s$ . So for the last 10 bits of the message, the sender can weaken the key at a linear rate of 1 key bit per packet bit. Therefore, only the first 10 bits of the key need to be scheduled. Thus, the message size becomes  $10 + \sum_{i=0}^9 2^i = 1033$  bits, which is a reasonable size. Note that if  $T_{\pm}$  allowed for only the transmission of a smaller number of bits, we can linearly weaken the key by more than one key-bit per transmitted bit. This slightly increases the computation cost of key inferring but only on a small number of bits. The revised key scheduling algorithm is illustrated in Figure 4.

Next, we present the efficient backward decoding algorithm, its computation complexity and briefly discuss the key establishment protocol under TREKS.

## 4. EFFICIENT BACKWARD-DECODING

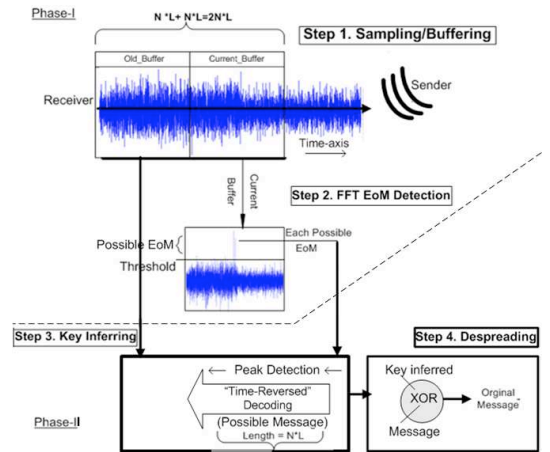


Figure 5: Workflow of TREKS Message Decoding

### 4.1 Overview of TREKS Decoding

MAC-masked TREKS enables efficient backward-decoding, which is best described as a two-phase phenomenon [See Figure 5]:

- **Phase-I:** Finding EoM by computing the cross-correlation between received chips and the PN-sequence generated with receiver's MAC address.
- **Phase-II:** Inferring the key in time-reversed fashion, which is used to despread the message.

## 4.2 Finding the EoM (Phase-I)

As shown in Figure 5, Phase-I consists of two steps, (a) sampling and buffering, and (b) FFT EoM detection. When new signal samples arrive, the receiver enqueues them into a FIFO. At any instance, the receiver only have to keep  $2nl$  chips in his buffer because after finding the EoM, he will have to traverse at most  $nl$  length before he recovers the message. We compute cross-correlation to achieve bit synchronization, a very common practice in SS systems [13]. However, calculating cross correlation is computationally expensive. We optimize this calculation (a) by using FFT, which reduces the cost of computing cross correlation from  $2n^2l$  to  $nl \log(nl)$ , and (b) by processing a batch of  $nl$  chips at once during FFT computation unlike conventional SS systems that process  $n$  chips (spread of a bit) at a time.

Symbol	Definition
$m$	message sent by the sender, as $z$ segments
$Seg[i]$	Segments of a message, where $1 \leq i \leq z$
$K[i]$	Key used to generate spread PN-sequence, $1 \leq i \leq z$
$K_i$	Possible set of keys, $1 \leq  K_i  \leq 2$ , that receiver tries to despread $Seg[i]$ with.
$S[i]$	Real-time signal that is sampled at the receiver side.
$PEoM[i]$	Array of possible EoM indices.
$M[i]$	Array of extracted complete messages.
$GetBuffer(.)$	Gets the next $n+l$ chips from the signal stream for sampling.
$DotProd(.)$	Dot product of two vectors (correlation function).
$FFT(.)$	Fast Fourier Transform.
$IFFT(.)$	Inverse Fast Fourier Transform.
$Fast\_Correlate(.)$	Calculating Convolution between a short and a long signal.
$Key\_Infer(.)$	Function to infer the key.
$Peak\_Detection(.)$	Function to detect peaks at $Seg[i]$ , $1 \leq i \leq z$
$Despread(.)$	Standard Spread Spectrum function to despread received signal.
$Signature\_Verify(.)$	Function to verify the sender.

Table 3: Additional notations

---

### Algorithm 2: Finding the End of the Message (EoM)

---

```

1. Old_Buffer = GetBuffer(S);
2. for each buffer of length (n * l) do
   Current_Buffer = GetBuffer(S);
   Set k = MAC_ADDRESS(Rcvr);
   Corr[1 : n * l] = Fast_Correlate(Current_Buffer, k);
   for each j in {1, ..., n * l} do
     If Corr[j] > threshold then
       push j into PEoM[];
   If PEoM[] is empty
     Old_Buffer = Current_Buffer;
   Else
     Buffer = concat(Old_Buffer, Current_Buffer);
     Key_Infer(Buffer, PEoM);

```

```

Fast_Correlate(Buff, key){
  Temp_Key[1:n*l] = Zeros;
  Temp_Key[1:n] = key;
  Input1 = FFT(Buff);
  Input2 = FFT(Temp_Key); //Pre-computed
  Corr[1:n*l] = IFFT(Input1*Input2');
  return Corr;
}

```

---

As illustrated in Algorithm 2, our FFT detection process iterates over each chip in the buffer to find the EoM. One challenge is that there might be more than one candidate for EoM, i.e., multiple values of the correlation vector may pass the threshold test to produce false positives. Thus, we enqueue all possible EoMs into  $PEoM[]$ , and pass it to Phase-II for further processing. We pick threshold value empirically by observing TREKS performance over large number of simulation runs, details of which is given in Section 5.

## 4.3 Message Extraction (Phase-II)

Phase-II consists of Step 3 and 4 as shown in Figure 5. In Step-3, we infer the key by finding the legitimate EoM out of

all PEoM found in Phase-I. For each PEoM, we begin time-reversed key inferring. For each key bit, we try two possible choices. Algorithm 3 shows this process. For a certain guess, if more than 50% of the total bits are detected in a schedule, then we confirm the value for this key bit and move onto the next. Otherwise, we abort the key inferring. Hence, we get Theorem 2.

**Theorem 2.** *The computational cost of TREKS message despreading is at most twice the computational cost of conventional SS systems with a pre-shared key.*

**Proof.** For each segment, the receiver attempts to despread the bits with two potential keys. Therefore each bit is despread twice. Leading to a computational cost of twice a conventional spread-spectrum. Note, that this cost can be reduced by eliminating one of the two keys after attempting only few bits of a packet.  $\square$

In Phase-II, another optimization we employ is that after we find the EoM, instead of computing FFT each time to synchronize with the bits of the message, we compute the dot-product between  $n$  chips and the PN-sequence. The abortion of key inferring process implies a packet loss otherwise we despread the message using the key inferred [Step-4]. We discuss the choice of the threshold values used in Algorithm 2 in Section 5.

---

### Algorithm 3: Message Extraction

---

```

1. Key_Infer(Buffer, PEoM){
  for each possible EoM j in PEoM do
    PeakPos = n+j; //EoM = Buffer[n+j]
    endIdx = PeakPos-n; //End of Seg[z-1]
    for each p in {1, ..., z} do
      startIdx = endIdx - |Seg[z]| + 1;
      CntOfSucc = 0;
      for each key candidate k in K_{z-p} do
        succ = Peak_Detection(k, Buffer,
                              startIdx, endIdx);
        CntOfSucc = CntOfSucc + succ;
      If (CntOfSucc == 1)
        K[p] = k;
      Else
        Abort Key_Infer(Buffer, PEoM);
    endIdx = startIdx;
  m = Despread(Buffer[j - (n * l) + 1, j], K[]);
  Enqueue m into M[];
}

2. Signature_Verify(M[]);

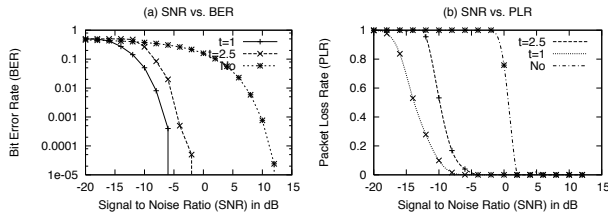
Peak_Detection(key, Buf, startIdx, endIdx)
{
  ExpNumofPeaks = (endIdx - startIdx)/n;
  CntOfPeaks = 0;
  for each d in {1, ..., ExpNumofPeaks} do
    P = DotProd(key, Buf[startIdx, startIdx+n]);
    If P > threshold then
      CntOfPeaks = CntOfPeaks + 1;
      startIdx = startIdx + (d * n) - 1;
  If CntOfPeaks > 50% * ExpNumofPeaks
    succ = 1;
  Else
    succ = 0;
  return succ;
}

```

---

## Signature Authentication and Key Establishment

At the end of Algorithm 3, depending on the type of jammer and its strategy, a receiver might end up recovering more than one message, namely the jammer messages. In that case, the receiver has to verify the sender using some



**Figure 6: Two graphs (a) SNR vs. BER (b) SNR vs. PLR.  $t = \frac{T_{threshold}}{average}$  and “No”  $\rightarrow$  without TREKS.**

kind of mutual authentication and identification mechanism. TREKS uses 160 bit Elliptic Curve based Digital Signature Algorithm (ECDSA) to authenticate the nodes and their data sent over the channel. A 160-bit ECC key provides the same level of security as that of a 1024-bit RSA key [8], which is sufficiently secure for the purposes of a session-based encrypted message transmission of TREKS. The choice of the key establishment protocol for TREKS need not be a specific key establishment protocol. Similar issue is already discussed extensively in [14]. So, we will use the same protocols of [14].

## 5. PERFORMANCE EVALUATION

We evaluate the performance of TREKS in terms of the Packet Loss Rate ( $PLR$ ) as a function of communication/jammer energy, computation cost, and storage cost. Based on Fact 1, we can focus on two jammers: (1) additive white gaussian jammer (whose energy is reduced by a factor  $n$ ) evaluated in Section 5.1, and (2) jammers spreading a signal with the receiver MAC address evaluated in Section 5.2. Without knowing the beginning of the transmission, the jammer is forced to operate as a memoryless jammer with a rate  $\lambda$ . We call these  $\lambda$ -jammers. Note that if  $\lambda = 1$ , it becomes a continuous jammer.

**Simulation Setup:** We use MATLAB to simulate the communication, jamming, and message extraction under various settings of the configurable parameters to depict different types of jammers under different scenarios. All the graphs are based on 10K simulation runs of same parameter setting. The variables of our simulations are:

Spreading Factor, $n$	100
Packet Size, $l$	1033 bits
Key Size, $n$	19
Jammer Power to Signal Power Ratio, $JSR$	[1..100]
Normalized Signal Power	0 dBW
Noise Power	-20 dBW

**Table 4: Parameters for Simulation.**

### 5.1 TREKS vs. Gaussian Jammers

We consider the case where the sender and the receiver communicate under a white Gaussian jammer. From Fact 1, this corresponds to interferers not using the destination MAC address. Their interference results in Gaussian noise of energy reduced by a factor  $n$ .

#### 5.1.1 Packet Loss Rate (PLR)

The  $PLR$  under our model implies one of the following: (a) Key Infer Failure, (b) EoM missing, and (c) High BER (over 15% [11]). Figure 6 shows the PLR and the BER in-

curred using TREKS, as a function of an increasing SNR and different detection threshold. Note that due to the imperfect synchronization and EoM recovery, we only obtain a gain of 15 – 17 dB (i.e., 20 to 50 times resiliency gain).

#### 5.1.2 False Positives

The number of False Positives (FP) encountered during the FFT EoM detection process affects the performance of TREKS in terms of its computational delay. In fact, we use the PLR and the number of FPs observed while running TREKS at a fixed noise level of 0dB to choose the peak detection threshold used in Algorithm-2.

We define threshold as  $t * avg$  where  $avg$  is the average of the correlation vector produced by `fast_correlate(.)` of Algorithm-2, and  $t$  is a multiplier. Based on the results from Tables 5 and 6, we chose  $t$  to be 2.5 because of a much smaller FP rate observed at threshold =  $2.5 * avg$ , even though we loose about 2dB of jammer resiliency.

SNR (dB)	t=1.0	t = 2.0	t = 2.3	t = 2.5	t = 3.0
-10	11.79%	1.48%	0.94%	0.58%	0.22%
-5	11.80%	1.48%	0.94%	0.58%	0.22%
0	11.79%	1.47%	0.96%	0.59%	0.22%
5	11.79%	1.51%	0.98%	0.61%	0.23%
10	11.78%	1.57%	1.01%	0.64%	0.25%

**Table 5: False Positives (FP)**

SNR (dB)	t=1.0	t = 2.1	t = 2.3	t = 2.5	t = 2.9
-10	19.00%	48.00%	49.50%	47.50%	64.50%
-5	0.00%	0.20%	0.50%	1.50%	4.00%
0	0.00%	0.00%	0.00%	0.00%	0.00%

**Table 6: Packet Loss Rate (PLR).**

**Important Observation:** Figure 7 shows that we detect almost all of the FPs among PEOms by the first two iterations (stages) of `key_infer()` in Algorithm 3 when threshold =  $2.5 * avg$ . Thus,  $FP$  does not impact TREKS computationally by much. The increase in computation cost is negligible compared to the decoding cost, which itself is less than double the cost of decoding in traditional SS.

#### 5.1.3 Computation Cost

Operation	Using GPU	Lab Computer
FFT benchmark	1ms	28ms
Key Inferring	-	1ms
Signature Verification	-	1ms

**Table 7: TREKS Computation Cost.**

Table 7 shows the computation cost of TREKS performed in our lab computer versus using a GPU NVidia GeForce 8800 GTX. Using the latter, we can accelerate the FFT computation by 28 times [9]. The specification of our lab computer is a 64-bit Intel(R) Core(TM)2 CPU 6400 @2.13GHz with 3GB memory. It clearly shows that with appropriate off-the-shelf hardware, TREKS can operate in real time with its total execution time under 3ms. We used OPENSSL-0.9.8 version to calculate the benchmark for verifying 160-bit ECC-DSA.

#### 5.1.4 Storage Cost

The storage cost of TREKS accounts for (a) the total number of messages recovered at the end of message extraction, and (b) the size of the FIFO used in buffering the signal

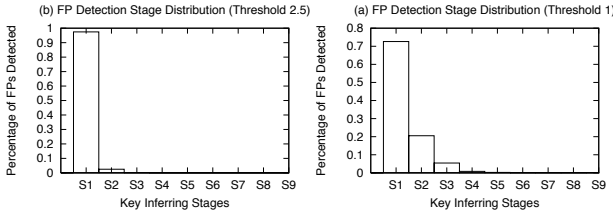


Figure 7: Distribution of the FP detection stage.

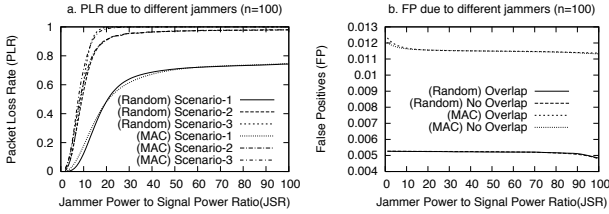


Figure 8: Jammer performance comparison.

samples in Algorithm-2. Even if a jammer injects  $j$  packets, we store at most  $(j + 1) * l/8$  bytes, and the *current\_buffer* in Algorithm-3 holds  $n * l$  samples. Hence, the storage cost of TREKS is  $4 * n * l + (j + 1) * l/8$  bytes (assuming each sample is a 32-bit I/Q value), clearly within the storage capacity of today’s computer.

## 5.2 TREKS vs. $\lambda$ -Jammers

Consider a discretized time with timeslots of duration  $n * l$  chips. We define two different kinds of jammers that take parameters  $\lambda$  and  $JSR$ .  $\lambda$  represents the probability that a jammer sends a jamming message at a given timeslot (this corresponds to discretization of a Poisson memoryless jammer to a Bernoulli jammer), and  $JSR$  is the jammer to signal power ratio. The cost of the jammer is  $\lambda * JSR$ , and its goal is to maximize the  $PLR$  for a given budget. In our simulation, we assume that the sender is always sending messages. Note that the actual jammer impact will be less than the simulation graph’s because the jammer does not know when a transmission occurs. Thus, a source transmitting with probability  $\mu$  would cause a jammer efficiency decrease by a factor of  $1/\mu$ .

**Jammer Types:** Jammers could also send partial messages but this can be independently addressed with appropriate interleaving and coding [11]. Hence, we consider following jammers in our simulation:

- **(Random) Jammer-1:** Inserts an  $l$ -bit message, each bit spread with a random PN-sequence.
- **(MAC) Jammer-2:** Inserts an  $l$ -bit message, each bit spread with the PN-sequence generated using the MAC address of the receiver as the seed.

**Jamming Scenarios:** Consider a data message that occurs inside a two timeslot (TS) window. Now, a jammer message might occur in the first, second, both or none of the timeslots. This gives rise to following possible scenarios:

- **Scenario-1:** Jammer message occurs in the first TS.  
*Impact:* Key inferring.
- **Scenario-2:** Jammer message occurs in second TS.  
*Impact:* EoM detection.

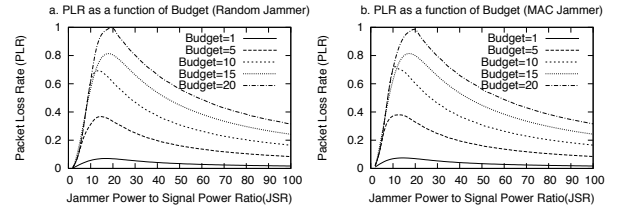


Figure 9: Jammer performance under fixed budget.

- **Scenario-3:** Jammer message intersects both TS.  
*Impact:* Key inferring and EoM detection.
- **Scenario-4:** Does not occur during those two TS.  
*Impact:* None.
- **Scenario-5:** Jammer’s packet is perfectly synchronized with the sender packet at the receiver side.  
*Impact:* If perfect synchronization was possible, then there is a 0.5 probability that the last bit of the message is jammed, hence causing to miss the EoM.

**Observation:** In Figure 8(b), we show that no matter which of the four scenarios we run, there is no incentive for the jammer to increase its  $JSR$  if its objective is to increase the FPs. With overlapping or without, the resulting number of FPs that impacts EoM detection is same.

For a given  $\lambda$ , let’s define the expected  $PLR$ :

$$E[PLR] = E_1 * \lambda(1 - \lambda) + E_2 * \lambda(1 - \lambda) + E_3 * \lambda^2 + E_4 * (1 - \lambda)^2$$

where  $E_1, E_2, E_3,$  and  $E_4$  are the expected  $PLR$  for above defined Scenarios-1,2,3 and 4 respectively.

Figure 8(a) shows that for both the MAC and Random jammer, scenario-2 has more impact on  $PLR$  than Scenario-1 and Scenario-3 has slightly more impact than the both. Obviously,  $E_4 = 0$ . Figure 9 shows that Random Jammer and the MAC Jammer attain their optimum (expected)  $PLR$  approximately when  $10 \leq JSR \leq 15$ . Note that these results are based on the assumption that the communication is always happening. In reality, the impact of the jammer will be much less.

**Case of the MAC Jammer:** The MAC jammer outperforms the Random jammer only in terms of the numbers of FP produced. However, Figure 7 shows that by the third stage of key inferring, almost all of the FPs are detected. Thus, its impact in terms of computation and delay is negligible compared to decoding cost. In terms of  $PLR$ , it is a very close race between the MAC jammer and the Random jammer with MAC jammer winning by a slight margin. This is simply because only the last bit of the message is spread with receiver’s MAC address.

**Case of Perfect Synchronization (Scenario 5):** We believe that it is very hard for the jammer to attain Scenario 5, i.e., achieve perfect synchronization, because under our mechanism the jammer does not know when the communication is happening, and only one (last) bit of the packet is actually spread with receiver’s MAC address. Therefore, the probability of Scenario 5 is  $1/n$ .

## 6. CONCLUSION AND FUTURE WORK

We introduce a method for achieving SS anti-jamming without a pre-shared key. Our method has zero energy overhead in comparison with conventional SS communication.

Our solution relies on intractable forward-decoding and efficient backward-decoding. We propose several algorithms to optimize the decoding and show that the computational cost of despreading is less than twice the conventional SS cost. Our method has additional benefits of delayed detection and destination-oriented transmission making jamming infeasible and keeping its impact to minimal by prohibiting jammers from simultaneously jamming multiple receivers.

**Future Work:** Since we focus on the key establishment for systems like SS, graceful degradation of the system throughput due to small PLR and intermittent losses won't affect our protocol. However, if we were to extend TREKS for long-lived communication without key establishment, then it would be interesting to investigate their impact. That is our future work. Furthermore, we believe that extending TREKS to today's popular systems, such as Wideband Orthogonal Frequency Division Multiplexing (W-OFDM), can increase the applicability of our scheme. We plan on studying different extensions of TREKS in future.

## 7. ACKNOWLEDGMENTS

This work was partially funded by NSF grant 0448330 (CAREER) and NSF CyberTrust grant 0716581.

## 8. REFERENCES

- [1] B. Awerbuch, A. Richa, and C. Scheideler. A jamming-resistant mac protocol for single-hop wireless networks. In *ACM PODC*, 2008.
- [2] E. Bayraktaroglu, C. King, X. Liu, G. Noubir, R. Rajaraman, and B. Thapa. On the performance of ieee 802.11 under jamming. In *Infocom*, 2008.
- [3] M. A. Bender, M. Farach-Colton, S. He, B. C. Kuzmaul, and C. E. Leiserson. Adversarial contention resolution for simple channels. In *SPAA*, 2005.
- [4] T. Brown, J. James, and A. Sethi. Jamming and sensing of encrypted wireless ad hoc networks. In *ACM MobiHoc*, 2006.
- [5] A. Chan, X. Liu, G. Noubir, and B. Thapa. Control channel jamming: Resilience and identification of traitors. In *IEEE ISIT*, 2007.
- [6] J. Chiang and Y.-C. Hu. Cross-layer jamming detection and mitigation in wireless broadcast networks. In *MobiCom*, 2007.
- [7] S. Gilbert, R. Guerraoui, and C. Newport. Of malicious motes and suspicious sensors: On the efficiency of malicious interference in wireless networks. In *OPODIS*, 2006.
- [8] B. Gupta, S. Gupta, and S. Chang. Performance analysis of elliptic curve cryptography for ssl. In *MobiCom*, 2002.
- [9] <http://www.cv.nrao.edu/pdemores/gpu/>. Gpu benchmarking.
- [10] M. Li, I. Koutsopoulos, and R. Poovendran. Optimal jamming attacks and network defense policies in wireless sensor networks. In *INFOCOM*, 2007.
- [11] G. Lin and G. Noubir. On link layer denial of service in data wireless lans. *Wireless Communication and Mobile Computing*, 2005.
- [12] K. B. Rasmussen, S. Capkun, and M. Cagalj. Secnav: Secure broadcast localization and time synchronization in wireless networks. In *MobiCom*, 2007.
- [13] M. K. Simon, J. K. Omura, R. A. Scholtz, and B. K. Levitt. *Spread spectrum communications; vols. 1-3*. Computer Science Press, Inc., NY, 1986.
- [14] M. Strasser, C. Popper, S. Capkun, and M. Cagalj. Jamming-resistant key establishment using uncoordinated frequency hopping. In *ISSP*, 2008.
- [15] P. Tague, D. Slater, G. Noubir, and R. Poovendran. Linear programming models for jamming attacks on network traffic flows. In *WiOpt*, 2008.
- [16] W. Xu, K. Ma, W. Trappe, and Y. Zhang. Jamming sensor networks: attack and defense strategies. *IEEE Network*, 2006.