Zero Pre-shared Secret Key Establishment in the presence of Jammers

Paper #1569170151

Abstract

We consider the problem of key establishment over a wireless radio channel in the presence of a communication jammer, initially introduced in [17]. 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. Our approach is based on the novel concepts of *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 applies *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 very efficient both in terms of resiliency against jammers and computationally. Finally, our technique has additional desirable features such as the inability to detect packet transmission by non-source nodes until the last few bits are being transmitted, and the destination-specific transmissions. To the best of our knowledge, this is the first solution that is optimal in terms of communication energy cost with little storage and computation overhead.

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 to malicious behavior is obviously of significant importance for military communication in a battle-field. It is also rapidely gaining significance in civilian and commercial applications due to the increased reliance on wireless networks for connectivity to the cyber-infrastructure, and applications that will monitor our physical infrastructure such as tunnels, bridges, landmarks, and buildings.

Jamming and anti-jamming techniques for the physical layer of wireless systems supporting mostly voice communication have been extensively studied for several decades [15]. However, it is only recently that the popularity of multi-hop data networks with more sophisticated medium sharing, coding, and application protocols opened the door for more 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, size of the congestion control window, etc. For example, by transmitting a few pulses at the right frequency, right time and right location, an tremendously energy/computation efficient attack can be implemented with off-the-shelf hardware [4, 5, 12, 13, 19, 21].

1.1 Motivation

Spread Spectrum (SS) is one of the most efficient and used mechanisms for building jamming-resilient communication systems. In [17], Strasser et al. recognized that the control mechanism of the underlying SS can be targeted. SS requires the communicating nodes to pre-share a secret key that is used to generate a cryptographically-strong pseudo-noise (PN) spreading sequence. In many scenarios (e.g., large number of dynamically associating/diassociating nodes) this shared key has to be established over an open channel. An adversary can therefore focus its jamming on the key establishment protocol. This problem was introduced as the anti-jamming/key establishment circular dependency problem [17]. 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 random spreading sequence

was used), it is too late for him 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 packet with a cryptographically-strong PN-sequence. The PN-sequence is derived from a random key whose entropy decreases as we get closer to the end of the message transmission (See Figure 4). 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. Forward-decoding the packet requires guessing the whole key initially, which is infeasible for the jammer to do (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, however, our scheme ensures that at each instant the time it takes for a node to brute-force the PN sequence plus the TX/RX turn-around 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 [17], is that it does not require more energy for transmitting packets. It is in fact as energy efficient as the conventional spread spectrum communication where the communicating nodes pre-share a secret key. UFH, on the other hand, requires on average n times more energy then traditional spread spectrum, where n is the spreading factor in the order of hundreds. We achieve this communication-energy efficiency by slightly increasing 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 3) and the storage required is of one packet length. A secondary advantage of our technique is delayed communication detection, which makes it practically impossible for an adversary to sense an on-going communication until it is "almost" over. This stealthiness forces an adversary to be an energy innefficient channel-oblivious jammer [2].

1.2 Related Work

Anti-jamming techniques were extensively studied for decades [15]. Most of the earlier mechanisms focussed on physical layer protection and made use of spread-spectrum techniques, directional antennas, and coding schemes. At the time, most wireless communication was not packetized, or 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 and thus, needs for more complex applications and deployment environments have emerged. Several specifically crafted attacks and counter-attacks were proposed for: packetized wireless data networks [12, 13], multiple access resolution in the presence of adversaries [1–3], multi-hop networks [12, 20, 21], broadcast communication [5, 6, 18], cross-layer attacks [13], and navigation information broadcast [14]. 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 [17]. They name this phenomenon as the anti-jamming/key establishment circular dependency problem.

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 at a significantly slower rate. 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 introduced the first reliable key establishment protocol for SS without a pre-shared secret. However, unlike SS systems with pre-shared keys, the proposed mechanism incurs an energy increase by a factor of n due to the required redundancy in packet transmissions (retransmissions of message fragments that are not received). This is the closest work related to our paper. Our mechanisms retain 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.

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 [1, 8]. These techniques are much less energy efficient then SS. Note that SS can still be used in narrow band if the signal is spread in time at no additional energy cost. The catch here is that the data rate is reduced by the spreading factor. The data rate reduction is not necessarily a limitation as two nodes can have multiple simultaneous communications as in Code Division Multiple Access systems. Thus, this mechanism has the potential of achieving an overall higher network throughput (see [7] for the theoretical derivation of the capacity region of CDMA systems). Our goal in this paper is to enable SS even in the absence of a pre-shared key.

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). This forces the jammer to become an energy-inefficient channel-oblivious jammers [2].
- A destination-oriented scheme that prevents efficient simultaneous-attacks of multiple receivers.
- Computationally efficient end of the message detection (a FFT-based technique), and message extraction (a key-scheduling algorithm requiring at most twice the cost of conventional spread spectrum decoding to guess the key and despread the packet).

2 Setup Model

In this section, we describe the basic communication model and the adversary model considered in our study.

2.1 System Model

We consider a wireless communication network where several nodes are trying to establish pairwise-shared secret keys that would enable SS communication. Our model and the problem formulation is very similar to [17]. We focus on a pair of communicating nodes along with a jammer, all sharing one radio-frequency channel. The jammer's objective is to prevent the establishment of a secret key between the communicating nodes, because once this key is established, the communicating nodes can use conventional SS making them resilient to jamming. Our main objective is to devise a jammer-resilient message-delivery mechanism with no pre-shared information. This mechanism will be used, by a Mutual Authentication and Key Agreement Protocol (MAKAP), to deliver few messages and establish a key for future SS communication. We consider the same MAKAP as [17], namely Elliptic Curve Diffie Hellman (ECDH), because of the small number of messages exchanged (i.e., 2) and their short length. Our method uses Direct-Sequence SS (DSSS), but it easily generalizes to Frequency-Hoping SS (FHSS).

Assumption:

- We assume that there exists a trusted Certificate Authority (CA) that issues digital certificates attesting each user's public key [1].
- Anything that is known to the receiver about the protocol and the sender is known to the jammer.

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 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 own jamming cost. We define jammer's performance as the trade-off function relating the packet loss rate with the total jamming cost. Our classification of the adversary attacks is inspired by the *active* attack categorization of [16] and the attacker model of [17]. 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 TREKS.

Assumption

- We ignore potential gains of configuring the physical layer parameters such as physical distance, antenna gains, and coding schemes. These parameters can be independently optimized.
- Our model does not consider the case where the jammer can block the propagation of the radio signal.
- 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 or cause failure of 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 (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. This attack is unlikely in a deterministic way, since it is infeasible for a jammer to detect on-going communication under our mechanism until last few bits of the message are sent.
4. **Denial of Service**: The attacker inserts partial or complete messages to overwhelm the receiver's (1) packet decoding, and (2) signature verification processes. Note that this is a stronger jammer then the replay jammer.

In Section 5, we will dicuss and analyze the impact of specific jammers tailored for our approach.

[1] 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.

3 Time-Reversed Message Extraction and Key Scheduling (TREKS) in DSSS

TREKS is a communication approach based on zero pre-shared key spread spectrum(ZPKS), specifically DSSS in this paper. We will first present the core idea of zero pre-shared key DSSS and its efficiency again jamming. Then we propose a novel key scheduling scheme, which enables efficient backward-decoding, and thus making TREKS optimal in terms of both communication energy cost and computation and storage cost.

3.1 Zero pre-shared key DSSS

Sender S, receiver R, and jammer J all share the same channel. Let M denote the message that S wants to transfer to R, l the length of M in bits. Prior to the start of transmission, S randomly generates a secret key K of k bits. Unlike conventional DSSS, K is not known to anyone but S when communication occurs. S uses K to generate a cryptographically strong PN-sequence and uses it to spread M. Although, PN-sequence cryptographically generated from keys (e.g., seeding a symmetric encryption algorithm such as AES or DES) are not optimal in terms or orthogonality, they have a very satisfactory performance and have been used in many military spread spectrum communications system [15].

In conventional DSSS, S and R pre-share the secret key. R keeps attempting to despread incoming signals with pre-shared key until she detects the beginning of the message, then she starts forward-decoding the whole message. In zero pre-shared key DSSS, R needs to first identify the key K chosen by S. Without knowing K, R does not even know when such DSSS communication occurs, so she needs to brute force all possible keys on each chip of the incoming signal until she finds a key that could properly decode the incoming signal. Given key size as k bit, the complexity (brute force) of exploring the key space is $O(2^k)$. Obviously, this is impractical for real-time communication when no information is available about the start of a packet. In Section 3.3, we show how backward-decoding with a key schedule makes our approach efficient for real-time communication.

3.2 Jamming resiliency

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

3.2.1 Communication energy efficiency

We present the way the packet data bits are spread and how the total energy per packet is preserved. We also show that the cost for the jammer to counter the effect of spreading requires an energy increase by a factor of n. Let us first introduce some terminology:

- $d \in \{+1,-1\}$: data bit being sent, both 0 and 1 are equally probable, otherwise the data can be compressed and might also be used by the adversary.
- $\hat{d} \in \{+1,-1\}$**:** estimated data bit on receiver side.
- n**:** Spreading factor.
- $pn_{i\in\{1,\ldots,n\}} \in \{-1,+1\}$**:** i^{th} chip of cryptographically designed spreading sequence unknown to the adversary.
- E_b**:** energy per transmitted bit (w.l.o.g, assume that we are sending one bit per unit of time).
- $u_i = d\sqrt{\frac{E_b}{n}}pn_i$**:** chip signals transmitted by sender. Note that the energy[2] per bit remains equal to E_b. We consider a Binary Phase Shift Keying modulation, but the results generalize to other modulations.
- J**:** jammer energy per unit of time.
- $I_{i\in\{1,\ldots,n\}}$**:** adversary's transmitted signals indexed at the chip level. The mean square of I_i is $\frac{J}{n}$ which corresponds to J amount of energy per bit.
- v_i**:** received signals indexed at chip level.
- $BER(E_b,J,m)$**:** Bit Error Rate at receiver side when sender is using E_b Joules per bit, adversary J Joules per bit, and transmitter spreading by factor m.

THEOREM 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. Let v_i denote the received signal indexed at the chip level:

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

where r_i is the jamming chip with unit mean square.
Consider the following decoding technique[3]:

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

The Bit Error Rate of the despread signal, $BER(E_b,J,n)$

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

[2]Energy is equal to the signal mean square.

[3]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.

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,

$$\begin{aligned} BER(E_b,J,n) &= \int_{n\sqrt{\frac{E_b}{J}}}^{\infty} \frac{1}{\sqrt{2\pi n}} e^{-\frac{x2}{2n}} dx \\ &= \int_{\sqrt{\frac{E_b n}{J}}}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x2}{2}} dx \end{aligned} \quad (1)$$

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. □

3.2.2 Computational infeasibility for jammer

In order to jam, in a cost efficient way, the adversary 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 then the packet transmission, even if the jammer eventually finds the key, he missed the chance to jam the transmission. We call this *intractable forward-decoding*, which is illustrated in Figure 1.

3.2.3 Limitations

Intractable forward-decoding is based on the fact of zero pre-shared secret, which also applies to the message-decoding process at the receiver. Since the receiver needs to try 2^k possibilities for spreading key on each incoming chip signal, it causes a considerably high computation overhead. This is a major limitation of the basic zero pre-shared key DSSS (ZPKS) scheme.

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 but still requires the same effort from an adversary to identify the key.

THEOREM 2. *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 to spread M with a k bits key, it is secure to spread the last $\frac{|M|}{2^i}$ bits with $k-i$ bit key, where $i \leq \log_2(|M|)$.*

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 bits key, we have

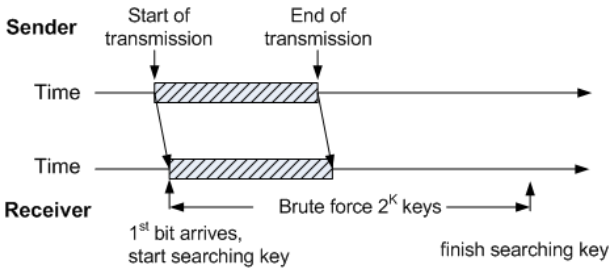

Figure 1. Message delivered before key is brute forced by adversary.

$$\begin{aligned} T_{trans}(|M|) &\ll T_s(k) \\ T_{trans}(\tfrac{|M|}{2}) &= \frac{1}{2} T_{trans}(|M|) \\ &\ll \frac{1}{2} T_s(k) = T_s(k-1) \end{aligned} \quad (2)$$

Eq. (2) shows that it is secure to encode $\frac{|M|}{2}$ 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 $T_{trans}(\frac{|M|}{2^i}) \ll T_s(k-i)$. Thus, it is secure to spread the last $\frac{|M|}{2^i}$ bits with $k-i$ bit key. □

The intuition behind Theorem 2 is that as transmission goes on, less time is left for jammer to find out the key, so it is safe to use a slightly weaker key to encode the rest of the message [4].

3.3.2 Spread key scheduling

Based on Theorem 2, we introduce a key scheduling scheme to TREKS. As shown in figure 2, instead of spreading the complete the 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 ith segment M_i is $\lceil \frac{|M|}{2^i} \rceil$. At the start of spreading process, we use full length key to spread M_1. Every time when we finish encoding one segment, we set the most significant bit of the key to a known value and resumes encoding the next segment with this 1-bit weaker key. We repeat this process until the last schedule, which is encoded with only 1 bit key. Here it is easy to see that the message length l has to be at least 2^k so that the key size k could be decreased to 1 bit as schedule goes on. For simplicity of presentation, we assume that $l = 2^k$ at this moment. We will show how to loose this constraint in later section. Algorithm 1 outlines the message segmentation and key scheduling scheme.

[4] Additional measures can be taken to prevent overlap between weakened key spaces.

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 1. Summary of the notations.

Algorithm 1: Sender encoding message with key schedule.

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

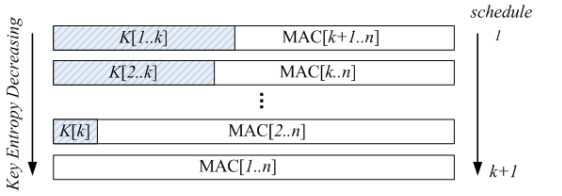

Figure 3. MAC-masked key scheduling.

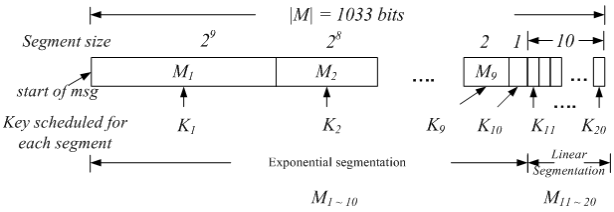

Figure 4. Key scheduling with linear tail.

Intractable forward-decoding: Based on theorem 2, the key size at each schedule is large enough for the size of corresponding message segment. Thus, 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, in order to detect the last message segment, the receiver just needs to attempt two keys. Once the receiver detects the potential end of message, he starts guessing the keys for previously received signals following the key scheduling scheme in reverse time. In order to guess the key for each previous schedule, receiver only needs to try two keys, because the bit difference for two adjacent schedules is only 1 bit. So the receiver needs to try up to $2*k$ keys to find out all the k key bits, which is significantly lower then the $O(2^k)$ guesses of the basic scheme.

3.4 Further improvements and discussion

3.4.1 MAC-masked key scheduling

In the key scheduling scheme presented above, the last scheduled key K_k is always either 0 or 1 for any sender/receiver pair. Hence, the jammer could jam with a PN-sequence generated by 0 or 1 key, which is likely to compromise the last message fragment. Once the end of the message 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

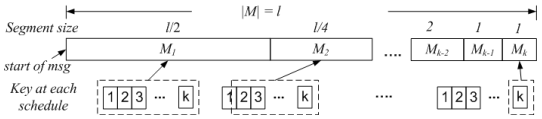

Figure 2. TREKS with key scheduling.

$i-1$ 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.

3.4.2 Key scheduling with linear tail

Consider a key size of $k = 20$, the jammer needs to explore 2^{20} keys. Even for a chip rate of 100Mcps (10ns chip duration), it is infeasible for a field deployed device to brute force the key transmission of few milliseconds (e.g., 1ms for 1000 bits spread with $n = 100$). However, 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 the 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 $T_{trans}(|M|) \leq T_\delta$, where T_δ 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 $10us$. Consider a spreading factor $n = 100$, chip rate of 100Mcps, then we have $T_{trans}(1) = 1us$. 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_δ allowed for only the transmission of a smaller number of bits, we can linearly weaken the key by more then 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

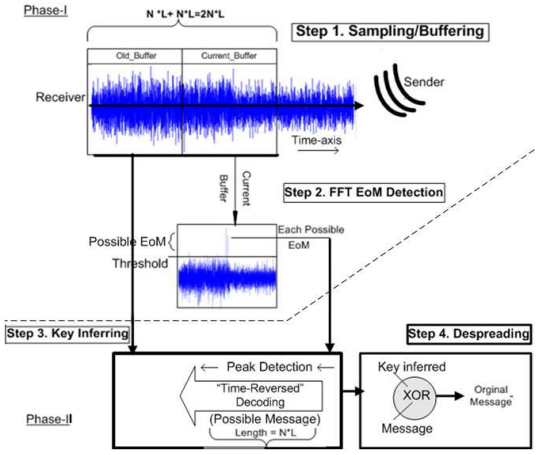

Figure 5. Workflow of TREKS Message Decoding

the key establishment protocol under TREKS.

4 Efficient Backward-Decoding

4.1 Overview of TREKSDecoding

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

- ***Phase-I***: Finding End of the Message(EoM) by computing the cross-correlation between received spread signal 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.

Symbol	Definition
m	message sent by the sender, as z segments
$Seg[i]$	Segments of a message, where $1 \le i \le z$
$K[i]$	Key used to generate spread PN-sequence, $1 \le i \le z$
K_i	Possible set of keys, $1 \le \lvert K_i \rvert \le 2$, 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], $1 \le i \le z$
$Despread(.)$	Standard Spread Spectrum function to despread received signal.
$Signature\_Verify(.)$	Function to verify the sender.

Table 2. Additional notations

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 $2n*l$ chips in his buffer because after finding the EoM, he will have to traverse at most $n*l$ length before he recovers the message. We compute cross-correlation to achieve bit synchronization, a very common practice in SS systems [15]. 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) process a batch of $n*l$ chips at once during FFT computation unlike conventional SS systems that process n chips (spread of a bit) at a time. 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.

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 ∈ {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;}
```

4.3 Message Extraction (Phase-II)

Phase-II consists of Step-3 and Step-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. At each stage of the process, we try two possible choices for the key-bit. Algorithm 3 illustrates this process. For a certain guess, if more than 50% of the total bits are detected in a segment, then we confirm the value at the corresponding key bit position and move onto the next. Otherwise, we abort the key inferring. Hence, we get Theorem 3.

THEOREM 3. *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 bits 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. □

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 ∈ PEoM do
    PeakPos = n+j; //EoM = Buffer[n+j]
    endIndx = PeakPos-n; //End of Seg[z-1]
    for each p ∈ {1,···,z} do
        startIndx = endIndx−|Seg[z]|+1;
        CntOfSucc = 0;
        for each key candidate k ∈ K_{z−p} do
            succ = Peak_Detection(k,Buffer,
                                  startIndx, endIndx);
            CntOfSucc = CntOfSucc + succ;
        If(CntOfSucc==1)
           K[p] = k;
        Else
           Abort Key_Infer(Buffer,PEoM);
        endIndx = startIndx;
    m = Despread(Buffer[j−(n∗l)+1, j], K[]);
    Enqueue m into M[];}

2. Signature_Verify(M[]);

Peak_Detection(key, Buf, startIndx, endIndx)
{
   ExpNumofPeaks = (endIndx - startIndx)/n;
   CntOfPeaks = 0;
    for each d ∈ {1,···,ExpNumOfPeaks} do
      P=DotProd(key,Buf[startIndx,startIndx+n]);
      If P > threshold then
          CntOfPeaks = CntOfPeaks+1;
      startIndx = startIndx+(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 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 [9], 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. The choice of the key establishment protocol is already discussed extensively in [17]. So, we will use the same protocols of [17].

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.

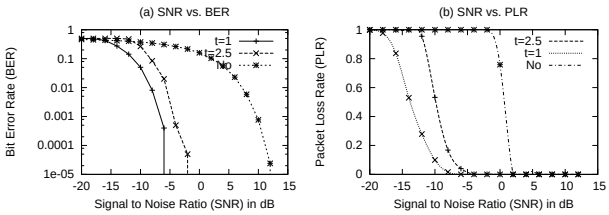

Figure 6. SNR vs. (a) BER (b) PLR with (Threshold = t*avg, t=1, t=2.5) and without (No) TREKS.

Based on Theorem 1, we can focus on two communication 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 destination MAC address evaluated in Section 5.2. Since, the adversary does not know the beginning of the transmission, he is forced to operate as a memoryless jammer with a rate λ. We use MATLAB to simulate the communication, jamming, and message extraction.

Simulation Setup: All the graphs are based on 10K simulation runs of each setting of parameters under our model. The variables that constitute our simulation 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 3. 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 Theorem 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% [13]). Figure 6 shows the PLR and the BER in TREKS system as a function of an increasing SNR and two detection thresholds. Note that due to the imperfect synchronization and EoM recovery, we only obtain a gain of $15-17$ dB (i.e., 20 to 50 resiliency gain).

5.1.2 False Positives

The number of False Positives (FP) encountered during the FFT EoM detection process directly impacts the performance of TREKS in terms of 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 the threshold as $t * avg$ where avg represents the average of the correlation vector produced by function fast_correlate(.) of Algorithm-2 and t represents the multiplier. Based on the results from Table 4 and 5,

we chose t to be 2.5 because of the much smaller FP rate even if 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 4. 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 5. Packet Loss Rate (PLR).

Figure 7 shows that we detect all of the FPs from PEoMs by the first iteration (stage) of the $key\_infer()$ in Algorithm 3 for threshold value of 2.5. Thus FP does not impact TREKS computationally by much.

5.1.3 Computation Cost

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

Table 6. TREKS Computation Cost.

Table 6 shows the computation cost of TREKS performed in our lab computer versus using a GPU NVidia GeForce 8800 GTX, we can accelerate the FFT computation by 28 times [10]. The specification of our lab computer is a 32-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 [11].

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 samples in Algorithm-2. Even if a jammer injects j packets, we only have to store at most $(j+1)*l/8$ bytes, and the size of $current\_buffer$ in Algorithm-3 is also $n*l$ samples (assuming two 16 bits -I, and Q- values per sample) only. Hence, the storage cost of TREKS is $4*n*l+(j+1)*l/8$ bytes, clearly within the realms of possibility with today's computer hardware.

5.2 TREKS vs. non-continuous Jammers

We consider a discretized time with timeslots of duration $n*l$ chips. We define two different kinds of jammers that take parameters λ and JSR. λ 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

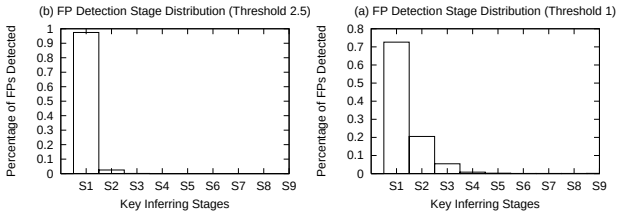

Figure 7. Distribution of the FP detection stage.

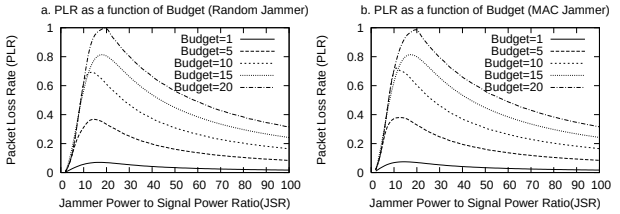

Figure 8. Jammer performance under fixed budget.

the jammer to signal power ratio. The cost of the jammer is $\lambda*JSR$, and his goal is to maximize the PLR for a given budget. Note that because the adversary does not know when transmissions are happening, the cost of the jammer should be further scaled by a factor μ which corresponds to the data transmission rate. In the following we consider the best case for the jammer where $\mu=1$. The jammer could also send partial messages but this can be independently addressed with appropriate interleaving and coding [13]. Hence, we consider following jammers:

- **(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.

Consider a data message; it randomly overlaps with two consecutive timeslots (TS). There are four possible scenarios based on if the first, second or both TS are jammed.

- **Scenario-1**: Only the first TS is jammed.
  Impact: Key inferring.
- **Scenario-2**: Only the second TS is jammed.
  Impact: EoM detection.
- **Scenario-3**: Both TS are jammed.
  Impact: Key inferring and EoM detection.
- **Scenario-4**: None of the TS are jammed.

Note that Scenario-4 implies no packet loss since there is no overlapping between the jammer and the data packet. Hence, we only show Scenarios-1,2, and 3 in Figure 9(a). In Figure 9(b), we compare Scenarios-3 and 4 and show that there is no incentive for the jammer to increase its JSR if its only objective is to increase the FPs. For a given λ (jamming rate), the expected PLR is

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

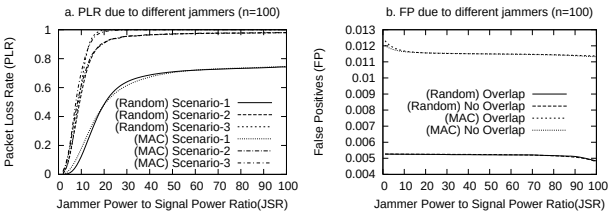

Figure 9. Jammer performance comparison.

where E_1, E_2, E_3,and $E_4(=0)$ are the expected *PLR* for above defined Scenarios-1,2,3 and 4 respectively.

Figure 8 shows the expected PLR for Scenarios-1,2,3 and 4. Depending on the budget, the jammer maximizes its impact on the receiver (PLR). We observe that Jammer-1 and Jammer-2 attain their optimum approximately when $10 \leq JSR \leq 15$. Even in the best case for the jammer ($\mu = 1$), the jammer needs to spend 10 times more energy to reduce the throughput to 30%. For $\mu = 0.1$, the jammer would need to spend 100 more energy then the communicating to reduce the throughput to 30%.

6 Conclusion

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 proposal relies on an intractable forward-decoding and efficient backward-decoding mechanism. We propose several algorithms to optimize the decoding and show that the computational cost of despreading is bounded by twice that of conventional SS communication. Our method has additional benefits such as making the communication undetectable until it is too late for an adversary to act. The proposed method is also destination-oriented preventing smart-jammers from simultaneously impacting multiple receivers.

7 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. Kuszmaul, 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] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 2006.

[8] 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.

[9] B. Gupta, S. Gupta, and S. Chang. Performance analysis of elliptic curve cryptography for ssl. In *MobiCom*, 2002.

[10] http://www.cv.nrao.edu/ pdemores/gpu/. Gpu benchmarking.

[11] http://www.openssl.org/. Openssl toolkit.

[12] M. Li, I. Koutsopoulos, and R. Poovendran. Optimal jamming attacks and network defense policies in wireless sensor networks. In *INFOCOM*, 2007.

[13] G. Lin and G. Noubir. On link layer denial of service in data wireless lans. *Wirel. Commun. Mob. Comput.*, 2005.

[14] K. B. Rasmussen, S. Capkun, and M. Cagalj. Secnav: secure broadcast localization and time synchronization in wireless networks. In *MobiCom*, 2007.

[15] 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.

[16] W. Stallings. *Cryptography and Network Security*. Prentice Hall, Inc., NJ, 2006.

[17] M. Strasser, C. Popper, S. Capkun, and M. Cagalj. Jamming-resistant key establishment using uncoordinated frequency hopping. In *ISSP*, 2008.

[18] P. Tague, M. Li, and R. Poovendran. Probabilistic mitigation of control channel jamming via random key distribution. In *PIMRC*, 2007.

[19] P. Tague, S. Nabar, J. Ritcey, D. Slater, and R. Poovendran. Throughput optimization for multipath unicast routing under probabilistic jamming. In *PIMRC*, 2008.

[20] P. Tague, D. Slater, G. Noubir, and R. Poovendran. Linear programming models for jamming attacks on network traffic flows. In *WiOpt*, 2008.

[21] W. Xu, K. Ma, W. Trappe, and Y. Zhang. Jamming sensor networks: attack and defense strategies. *IEEE Network*, 2006.