Efficient Spread Spectrum Communication without Pre-shared Secrets

Aldo Cassola, Tao Jin, Guevara Noubir, Bishal Thapa

Abstract—Spread spectrum (SS) communication relies on the assumption that some secret is shared beforehand among communicating nodes in order to establish the spreading sequence for long-term wireless communication. Strasser et al. identified this as the circular dependency problem (CDP). This problem is exacerbated in large networks where nodes join and leave the network frequently, and pre-configuration of secrets through physical contact is infeasible. In this work, we introduce an efficient and adversary-resilient secret sharing mechanism based on two novel paradigms (*intractable forward-decoding*, *efficient backward-decoding*) called Time Reversed Message Extraction and Key Scheduling (TREKS) that enables SS communication without pre-shared secrets. TREKS is four orders of magnitude faster than previous solutions to the CDP. Furthermore, our approach can be used to operate long-term SS communication without establishing any keys. The energy cost under TREKS is provably optimal with minimal storage overhead, and computation cost at most twice that of traditional SS. We evaluate TREKS through simulation and empirically using an experimental testbed consisting of USRP, GNU Radio, and GPU-equipped nodes. Using TREKS under a modest hardware setup we can sustain a 1Mbps long-term SS communication spread by a factor of 100 (i.e., 100 Megachips per second) over a 200MHz bandwidth in real-time.

Index Terms—Spread Spectrum, Zero Pre-Shared Secret, Anti-Jamming, GNURadio, USRP, Experimentation.

1 Introduction

Radio-Frequency (RF) wireless communication is exposed to adversaries that could eavesdrop trying to get hold of important information, or jam trying to prevent communication from happening. Resiliency to malicious behavior is highly desirable on wireless environments, as reliance on wireless communication becomes increasingly common for monitoring physical infrastructure or equipment.

Privacy issues in communication have been at the forefront of networks research for decades, focusing mostly on the protection of voice communication [2] and physical layer, and use of Spread Spectrum (SS) techniques. Networks were small, allowing for easy node pre-configuration. Today however, communicating nodes from various manufacturers enter and leave the networks dynamically, making pre-configuration impractical. The alternative of establishing Spread Spectrum keys over the air is subject to jamming, creating a Circular Dependency Problem (CDP as coined by [3]), an obstacle for wider Spread Spectrum deployments.

In this paper, we present an efficient and adversary-resilient secret sharing mechanism that that solves the CDP called Time Reversed Message Extraction and Key Scheduling (TREKS), based on two novel paradigms (*intractable forward-decoding*, *efficient backward-decoding*). This work, in conjunction with its preliminary version appearing in [1], provides the following key properties that, to the best of our knowledge, are not provided by any other system:

- No energy usage overhead compared to SS with pre-shared keys.
- Undetectable communication until the end of transmission forcing the jammer to become energy-inefficient and channel-oblivious [4].
- A destination-oriented scheme that prevents simultaneous attacks from a jammer targeting multiple receivers at once.
- Computationally efficient end of the message detection and message extraction.
- No receiver synchronization required.
- Computation cost at most twice of conven-

- *A. Cassola, T. Jin, G. Noubir, and B. Thapa are with CCIS, Northeastern University.*

A preliminary version of this material appeared at the 10th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'09) [1]. Research partially supported by NSF awards CNS-0959584 and CNS-0915985.

tional SS decoding.

- *Real-time* SS communication at 1Mbps spread by a factor of 100, i.e., 100 Megachips per second over a 200MHz bandwidth, a *four orders of magnitude* speed improvement over the initial UFH [3] solution with proposed parameters, and one order of magnitude latency improvement over our earlier estimate [1].
- No need for key establishment.

Our performance improvement comes through a new design and a combination of encoding and decoding optimizations. First, the intractable forward- and efficient backward-decoding lead to a powerful and resilient method for secret sharing. Second, the proposed implementation techniques, including block processing, FFT-based message detection, synchronization recovery, and key scheduling, play a vital role in optimizing the communication and decoding costs, making it fit for long-term and real-time communication.

1.1 Related Work

Anti-jamming techniques have been studied for decades, but reliable communication in the presence of adversaries has gained significant interest in the last few years. Several specifically crafted attacks and counter-attacks were proposed for packetized wireless data networks [5], [6], multiple access resolution in the presence of adversaries [7], [4], [8], multi-hop networks [9], [10], [6], broadcast communication [11], [12], [13], cross-layer attacks [5], 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. Recently, several countermeasures that do not consider the possibility of using SS were proposed, considering narrow RF bands or no pre-shared key [15], [8]. While some of these techniques are theoretically optimal for the considered physical layer, they are less energy efficient than SS. Strasser et al. recognized this pre-sharing requirement 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 [3]. They call this phenomenon the anti-jamming/key establishment CDP.

To solve the CDP, 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, sending 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, and 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 retransmissions. This is the closest work related to our paper.

Also, Strasser et al. [16] and Slater et al. [17] have proposed several coding-based mechanisms for error-detection in fragments and erasures-correction to improve UFH performance. Popper et al. propose a generalization of UFH for broadcast DSSS [18] but do not address the energy efficiency problem.

This paper is organized as follows. In Section 2, we present the system and the adversary model of TREKS. In Section 3, we present the main scheme. In Section 4, we present algorithms and optimizations for efficient message decoding. Section 5 evaluates TREKS using MATLAB simulations and experimentally on our test-bed, demonstrating TREKS's capability to perform real-time zero-preshared SS communication in the presence of various types of jammers. Finally, we conclude and discuss future work in Section 6.

2 System Model

Our model considers systems that are traditionally capable of performing SS communication, such as mobile ad hoc networks. Communicating nodes share a medium with the adversary. In this section we describe the goals, types of the participants, and assumptions we make in this paper.

2.1 Communication and Adversary

We consider a wireless network where nodes communicate in pairs in the presence of adversaries through the use of Spread Spectrum. Participants lack any pre-shared secret, which is a prerequisite

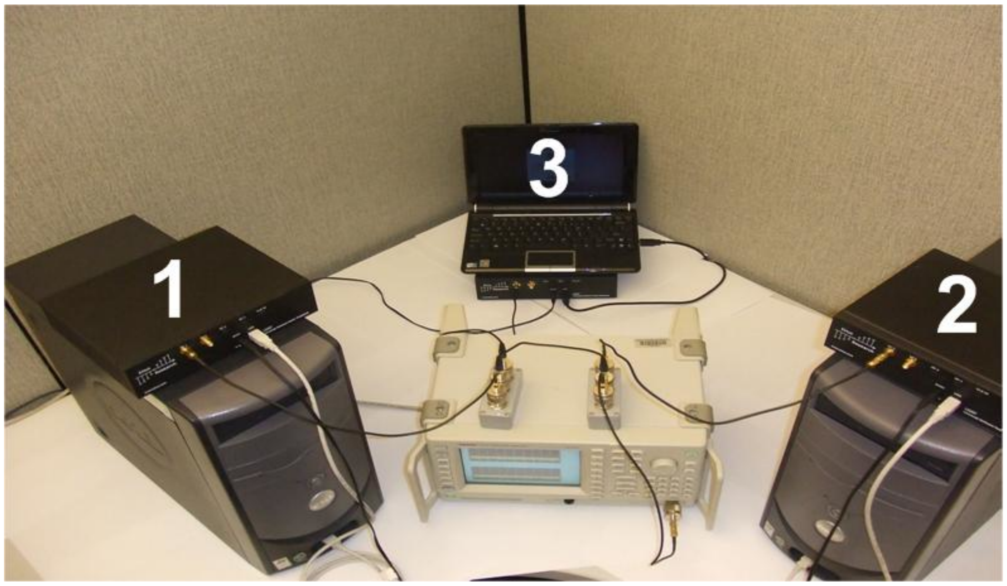


Fig. 1: Testbed for TREKS BER/PLR Experimental Evaluation: 1. USRP-equipped Sender node, 2. USRP-equipped Receiver node, 3. USRP-equipped Jammer node, 4. Nodes connected through Splitters and RF-Cable

for traditional SS communication systems. The goal of the sender is to establish an adversary-resilient and energy efficient communication. The only goal of the adversary is to prevent the receiver from decoding its messages.

Under our model, an adversary is within range of the sender and the receiver, and can possibly jam, replay previously collected messages or insert/modify bits of messages. The goal of the adversary is to prevent a successful reception of the message. The adversary's utility function is a trade-off between the energy cost spent on adversarial attacks versus the packet loss rate on the receiver side. We also evaluate the adversary in terms of the delay incurred by its attacks on the receiver's decoding process.

In this work we consider the following kinds of attacks by the adversary:

1) Jamming: The adversary may jam ongoing communication either reactively or obliviously sending a high power pulse. The jamming could be periodic, continuous or memoryless. The goal is to distort enough bits to cause packet decoding failures at the receiver.
2) Replay Attack: The adversary may insert previously collected messages to either cause a Denial of Service attack or simply a delay in the message extraction process at the receiver.
3) Modification: The adversary may target few bits in the message to modify its contents. We will show why this kind of attack is not feasible under TREKS because the jammer is unable to detect transmission until the last bit of the packet is emitted.
4) Insertion: The adversary may insert partial or complete messages following TREKS to overwhelm the decoding process at the receiver.

In Section 5, we implement protocol-specific adversarial strategies that are feasible in real-time within the limitations of the hardware available. We demonstrate the optimal adversarial strategy and establish its cost-efficiency against our scheme.

2.2 Assumptions

Under our communication model, the sender, receiver and adversary share the same channel, as well as information such as MAC addresses, key lengths, communication protocol, and encoding/decoding schemes. The seed of a cryptographic Pseudorandom Number (PN) generator that produces sequences to spread the signal is the only piece of information exclusively known to the sender. We also assume that the adversary cannot relay the brute-forcing of the key to some remote location with supercomputational power, and get the cracked key back within 1ms.

We ignore the gain obtained by configuring the physical layer parameters such as coding, and antenna gains, since they can be optimized independently of our mechanism. In addition, we do not consider the case where the adversary can completely block the propagation of the radio signal from the sender to the receiver. If the adversary has unlimited power and continuously jams the channel with a strong signal, then it can obviously reduce the throughput to zero, just like it would on traditional SS.

3 Time-Reversed Message Extraction and Key Scheduling

We first present the core idea of zero pre-shared key DSSS and its efficiency against jamming. Then we present our 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 share the same physical channel. Let M denote the message from S to R, and l the length of M in bits. Prior to the start of transmission, S randomly chooses a secret key K of length k bits. S then uses K to generate a cryptographically strong PN-sequence to spread M. Although PN-sequences generated from cryptographic means (such as AES or DES) are not orthogonally optimal, they have been used successfully in military spread spectrum communication systems [2].

We make the following assumptions about the nature of the data:

- Unlike conventional DSSS, the key K is not known to anyone but S when transmission starts.
- Both the length of message l and key length k are public information.
- The bits in M are 0 or 1 with equal probability. If they are not, they can be compressed.
- The details about our scheme, including the underlying algorithm for PN generation are public information.

3.2 Jamming resiliency

We first demonstrate the fundamental strengths of the proposed approach in terms of energy efficiency against jammers and key recovery intractability during transmission.

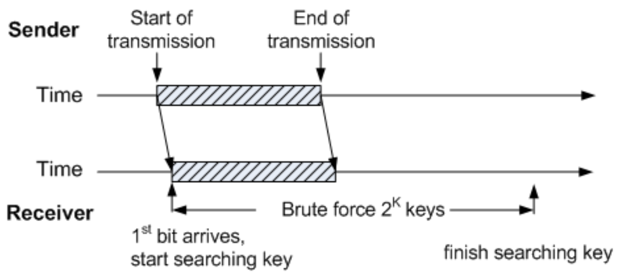


Fig. 2: Message delivered before key is found by adversary

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 sent by S. Both $+1$ and -1 are equally probable. Otherwise the data may be compressed to prevent the adversary from using this information.
- $\hat{d} \in \{-1, +1\}$: estimated data bit on receiver side.
- n: Spreading factor.
- $pn_{i\in\{1,\dots,n\}} \in \{-1, +1\}$: i^{th} chip of cryptographically designed spreading sequence unknown to the adversary.
- $r_{i\in\{1,\dots,n\}} \in \{-1, +1\}$: i^{th} chip transmitted by adversary.
- E_b: energy per transmitted bit. One bit is sent per unit time.
- $u_i = d\sqrt{\frac{E_b}{n}}pn_i$: chip signals transmitted by sender. Note that the energy (the signal mean square) 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.
- $I_{i\in\{1,\dots,n\}}$: 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.

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 to 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, but the result still holds in the general case. Let v_i denote the received signal indexed at the chip level:

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

Consider the following decoding technique[1]: $\hat{d} = 1$ iff $\sum_{i=1}^{n} v_i pn_i > 0$ The Bit Error Rate of the despread signal $BER(E_b, J, n)$

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

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,

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

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

1. We assume the receiver knows the bit synchronization. Section 4 shows how we achieve it.

3.2.2 *Computational infeasibility for jammer*

In order to efficiently jam, the adversary needs knowledge of the key K. If it is chosen uniformly at random, and its length k is chosen such that the time required to find the key is significantly higher than the packet transmission time, then the jammer will miss the chance to jam the transmission even if he eventually finds the key. We call this *intractable forward-decoding* (Figure 2.)

While the intractable forward-decoding prevents a jammer from detecting and acting on a transmission, it also makes the decoding for the receiver equivalent to that of the jammer if done naively. In the following section, we introduce the concept of key scheduling for *efficient backward-decoding*, which reduces computation complexity for the receiver from $O(2^k)$ to $O(2k)$ while keeping jamming resiliency.

3.3 Key scheduled reverse-time decoding

We introduce a key generation scheme for TREKS to spread data with a series of successively weaker keys. First we define the concept of a Key Schedule. Later we show a particular Key Schedule that protects against brute force search of the key. Finally we explore how key schedules may be constructed to protect against faster key searches.

3.3.1 *Key scheduling*

A sequence of keys $\{K_1, K_2, \ldots, K_k\}$ is called a *schedule* if every K_i is derived from K by setting its $i - 1$ most significant bits to the $i - 1$-most significant bits of some arbitrary value C. That is: $K_1 = K$; $K_i[1, \ldots, i-1] = C[1, \ldots, i-1]$; and $K_i[i, \ldots, k] = K[i, \ldots, k]$ where $K[1]$ is the most significant bit of K. In this instance we say that K is masked by C.

To spread (Figure 4) we partition the message into k segments M_i of size $|M_i|$. Each segment is spread with a PN sequence derived cryptographically from K_i where $i = 1, 2, \ldots, k$.

For instance, let KS_{BF} be a schedule that partitions the message in segments of size $\left\lceil \frac{|M|}{2^i} \right\rceil$ for $i = 1, 2 \ldots k$. Here it is easy to see that the message length $|M|$ has to be greater than 2^k so that the key size k can be decreased to 1 bit as the schedule goes on. For simplicity of presentation, we assume that $l = 2^k$. We will show how to loosen this constraint in a later section. Figure 3 outlines the message segmentation and key scheduling to spread a message with KS_{BF}.

Symbol	Definition
M	message to be transferred
K	secret key
l	length of message in bits
k	size of secret key in bits
$K[m \dots n]$	substring of K from m^{th} to n^{th} bit
$M[m \dots n]$	substring of M from m^{th} to n^{th} bit
K_i	ith key in schedule
M_i	message segment spread using K_i
N_i	leftover message after using K_i
$PN(\cdot)$	Secure PN-generating function

TABLE 1: Summary of the notation

```
**procedure** SENDER($M, K$)
    $N_1 \leftarrow M$
    **for** $i = 1, \dots, k$ **do**
        $K_i[i, \dots, k] \leftarrow K[i, \dots, k]$
        $K_i[1, \dots, i-1] \leftarrow C[1, \dots, i-1]$
        $M_i \leftarrow N_i\left[1, \dots, \left\lceil \frac{|N_i|}{2} \right\rceil\right]$
        $PN_i \leftarrow PN(K_i)$
        Spread $M_i$ with $PN_i$
        $N_{i+1} \leftarrow N_i[|M_i| + 1, \dots, |N_i|]$
    **end for**
**end procedure**
```

Fig. 3: Spreading message with key schedule

3.3.2 *Key size vs. Jamming resiliency*

For KS_{BF} we show that the effort required by the adversary remains the same as if the key entropy were kept constant. Theorem 1 describes what we mean by *same* effort precisely.

Theorem 1: Let $T_{trans}(l)$ denote the transmission time of l bits, and $T_s(k)$ the brute force time to find a k-bit key. Under our TREKS model, given a message M and a key of size k, if it is secure to spread M with a k-bit key, then it is secure to spread the last $\frac{|M|}{2^i}$ bits with a $k - i$ bit key, where $i \leq \log_2 l$.

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

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

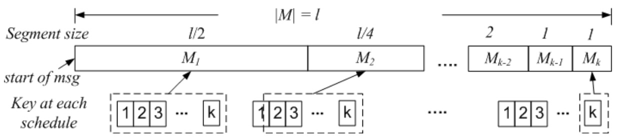


Fig. 4: TREKS with KS_{BF} key scheduling

The identity $\frac{1}{2}T_s(k) = T_s(k-1)$ holds because bruteforcing time is reduced by half when the key entropy decreases by one bit. Equation (2) gives the base case, implying that it is secure to encode $\frac{|M|}{2}$ bits with a $k-1$ bit key. By induction, it is then easy to show that $T_{trans}\left(\frac{|M|}{2^i}\right) \ll T_s(k-i)$. Hence, we prove that it is secure to spread the last $\frac{|M|}{2^i}$ bits of the message M with a $k-i$-bit key. $\square$

Intuitively, Theorem 1 states that as transmission proceeds towards the end of the message, there is less time for the jammer to find the key. Thus, it is safe to use slightly weaker keys to encode the bits towards the end as long as the spreading key is large enough to make its recovery longer than T_{trans}. Therefore, even as key entropy decreases, KS_{BF} protects every segment against bruteforcing, and thus the entire message.

3.4 Further improvements

3.4.1 *MAC-masked key scheduling*

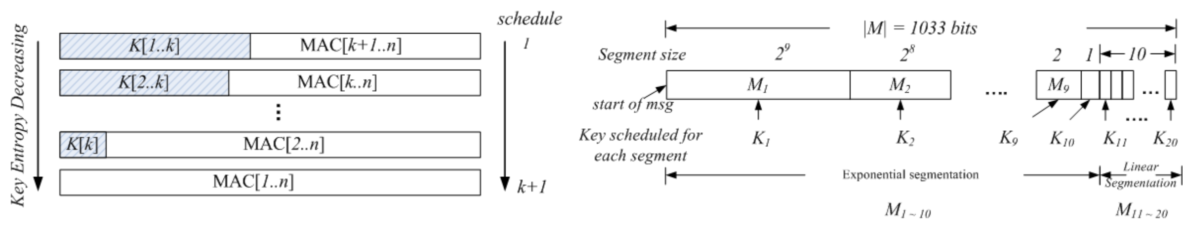
Because the value of C is public information, the jammer may use it to spread its signal and jam the last bit of the packet (the best jamming strategy for this scheme, as shown in Section 5.) To avoid this, the sender can use the destination's MAC address as the value of C, as illustrated in Figure 5, forcing the jammer to target one receiver at a time.

3.4.2 *Key scheduling with linear tail*

Section 3.3.1 assumed for simplicity that the length of the message $l = 2^k$ so that key size can be decreased down to 1 bit by the kth key in the schedule. This restriction is impractical even for small key sizes (e.g. for $k = 20$, packet size would be $1M$.) To relax the requirement, we observe that if $T_{trans}(|M|) \leq T_\delta$ (the radio turn-around time of the jammer), it is impossible for the jammer to jam M. For instance, consider a spreading factor $n = 100$, key size $k = 20$, chip rate of 100Mcps in 802.11, where $T_\delta = 10\mu$s. 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. Using this fact, only the first 10 message segments have exponentially decreasing size, reducing the total size of the packet to $10 + \sum_{i=0}^{9} 2^i$ = 1033 bits. The revised key scheduling algorithm is illustrated in Figure 5.

3.4.3 *Protection against faster key searches*

Building on the concepts of linear tailing and Theorem 1, it is possible to devise key schedules to



(a) MAC-masked key scheduling

(b) Linear Tail key Scheduling

Fig. 5: TREKS Optimizations

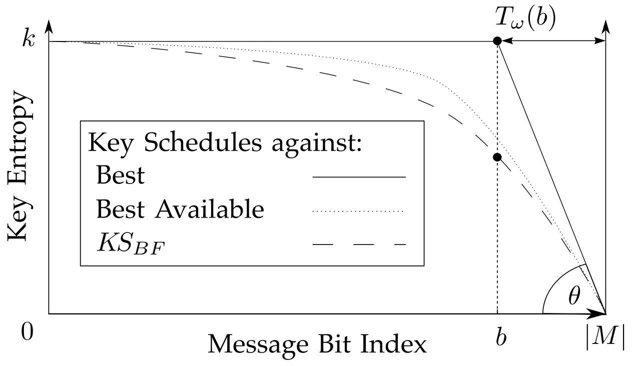


Fig. 6: Different key search profiles

Symbol	Definition
$\{K_i\}$	Key Schedule, $1 \leq i \leq k$
K_i	Key in schedule, $1 \leq i \leq k$
$\hat{K}_i$	Key guesses, $1 \leq \lvert\hat{K}_i\rvert \leq 2$ tried in despreading M_i
S_i	Signal sampled at R at time i.
$PEoM\,[i]$	Set of possible EoM indices
$E[i]$	Set of extracted messages
$GetBuffer\,(\cdot)$	Gets the next nl chips from signal stream
$DotProd\,(\cdot)$	Correlation of two vectors
$\mathcal{F}\,(\cdot)$	Fast Fourier Transform
$\mathcal{F}^{-1}\,(\cdot)$	Inverse Fast Fourier Transform
$FastCorrelate\cdot$	Function to convolute signals
$KeyInfer\cdot$	Function to infer the key
$PeakDetection\cdot$	Function to detect peaks at M_i, $1 \leq i \leq k$
$Despread\cdot$	Function to despread SS signal

TABLE 2: Notation for Efficient Backward Decoding

protect against better-than-bruteforce jammers (see Figure 6) by choosing a different message partition and entropy reduction rate. By selecting an entropy k for each bit b such that $T_\omega\,(b) < T_s\,(k)$ where T_s and T_ω are the given key search time and time until the end of message respectively, we can ensure the transmission is protected. In Figure 6, the area between schedules for Best and Brute Force searches may contain other schedules based on the jammer in play and its search time. The angle θ at which the entropy decreases, determines the effort for the receiver key recovery: $O\left(\frac{2^\Delta}{\Delta}k\right)$, where Δ is the entropy change in bits between schedules. For instance, a change of entropy of three bits each step forces the receiver to guess eight keys per key bit.

We believe KS_{BF} to be sufficient for protecting cryptographically secure PN generators such as ones based on AES-128 [19] since their best known cryptanalyses are close to brute force [20].

4 Efficient Backward-Decoding

In *Efficient Backward-Decoding* the receiver can deduce the key, due to the decreasing key entropy, by guessing two keys to find the end of transmission. Then, he successively tries 2 keys per segment in reverse time for a total of $2k$ guesses to retrieve the key, compared to the $O\left(2^k\right)$ of a brute force.

Because the receiver does not have the same time constraints as the jammer, he can store the received signals, and then process them backwards in time. TREKS is a two-phase procedure (see Figure 9). Phase I consists of finding the End of the Message (EoM) by computing the cross-correlation between the received spread signal and the PN-sequence generated with the receiver's MAC address. In Phase II the receiver infers the key in reverse time, starting where the high correlation was detected in Phase I. If correlation is maintained in time, then the key has been found, and the message is despread. The phases are summarized in Algorithms 7 and 8, based on the notation introduced in Table 2.

4.1 Finding the EoM (Phase-I)

Figure 9 depicts both steps of Phase I: sampling and buffering, and FFT EoM detection. When new signal samples arrive, the receiver enqueues them

```
procedure RECEIVER($S$, $R, n, l$)
    $OldBuffer \leftarrow GetBuffer\,(S)$
    $a \leftarrow MACAddress\,(R)$
    for all $CurrentBuffer \leftarrow GetBuffer\,(S)$ do
        $Corr\,[1, \ldots, nl] \leftarrow FastCorrelate\,(CurrentBuffer, a)$
        for all $j \in \{1, \ldots, nl\}$ do
            if $Corr[j] > threshold$ then
                push $j$ into $PEoM[]$
            end if
        end for
        if $PEoM[]$ is empty then
            $OldBuffer \leftarrow CurrentBuffer$
        else
            $Buffer \leftarrow concat\,(OldBuffer, CurrentBuffer)$
            $KeyInfer\,(Buffer, PEoM)$
        end if
    end for
end procedure
```

```
procedure FASTCORRELATE($Buffer$, $key$)
    $TempKey\,[1, \ldots, nl] \leftarrow$ Zeros
    $TempKey\,[1, \ldots, n] \leftarrow key$
    $FBuf \leftarrow \mathcal{F}\,(Buffer)$
    $Fkey \leftarrow \mathcal{F}\,(TempKey)$                          ▷ Pre-computed
    $Corr\,[1, \ldots, nl] \leftarrow \mathcal{F}^{-1}\,(FBuf \cdot Fkey)$
    return $Corr$
end procedure
```

Fig. 7: End of Message Detection

```
procedure KEYINFER($Buffer$, $PEoM[]$)
    for all $j \in PEoM[]$ do
        $PeakPos \leftarrow n + j$                       ▷ $EoM = Buffer[n + j]$
        $endIndx \leftarrow PeakPos - n$                     ▷ End of $M_{j-1}$
        for all $p \in \{1, \ldots, k\}$ do
            $startIndx \leftarrow endIndx - |M_i| + 1$
            $CntOfSucces \leftarrow 0$
            for all candidate key $c \in \hat{K}_{k-p}$ do
                $success \leftarrow$
                  $PeakDetection\,(c, Buffer, startIndx, endIndx)$
                $CntOfSuccess \leftarrow CntOfSuccess + success$
            end for
            if $CntOfSuccess = 1$ then
                $K_p \leftarrow c$
            else
                abort
            end if
            $endIndx \leftarrow startIndx$
        end for
        $m \leftarrow Despread\,(Buffer\,[j - (nl) + 1, \ldots, j]\,, \{K_i\})$
        Enqueue $m$ into $E[]$
    end for
end procedure
```

```
procedure PEAKDETECTION($key$, $Buffer, startIndx, endIndx$)
    $ExpNumOfPeaks \leftarrow (endIndx - startIndx)\,/n$
    $CntOfPeaks \leftarrow 0$
    for all $d \in \{1, \ldots, ExpNumOfPeaks\}$ do
        $x \leftarrow$
        $DotProd\,(key, Buffer[startIndx, \ldots, startIndx + n])$
        if $x > threshold$ then
            $CntOfPeaks \leftarrow CntOfPeaks + 1$
        end if
        $startIndx \leftarrow startIndx + (nd) - 1$
    end for
    if $CntOfPeaks > 0.5 \times ExpNumOfPeaks$ then
        $success \leftarrow 1$
    else
        $success \leftarrow 0$
    end if
    return success
end procedure
```

Fig. 8: Message Extraction

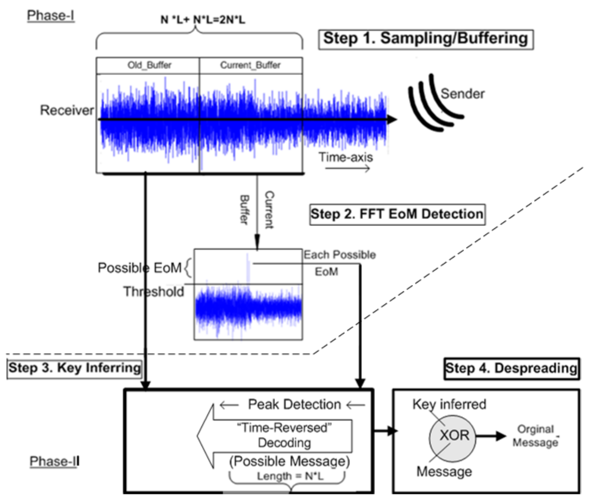


Fig. 9: Overview of TREKS Decoding

into a FIFO. At any instance, the receiver only has to keep $2nl$ chips in his buffer because after finding the EoM, he will have to traverse at most nl chips before he recovers the message.

To find the EoM and achieve bit synchronization the receiver computes the correlation between the signal and the expected PN sequence, a common practice in SS systems [2]. However, since calculating the cross-correlation is computationally expensive, we optimize this calculation by using a Fast Fourier Transform (FFT), reducing cost from $O\left(2n^2 l\right)$ to $O\,(nl \log nl)$ for every sample block of size nl (see Figure 7). The process iterates over the resulting buffer, checking for values exceeding a given correlation threshold. All vectors exceeding the threshold (even false positives) enter Phase II for further processing.

4.2 Message Extraction (Phase-II)

Phase II consists steps 3 and 4 of Figure 9. In Step 3 we infer the key, and find the actual EoM. For each candidate EoM, we begin the time-reversed bitwise key inferring. At each step in the process, we try two possibilities for the current key bit (Figure 8 specifies the process.) For each bit guess, we count the number of peaks seen in the segment. If the number of peaks is greater than half the number expected bits of the segment, we assume the value at the corresponding key bit position to be correct and move onto the next. Otherwise, we abort the key inferring, implying a packet loss. Once the key has been inferred, we despread the message. From this procedure we derive Theorem 2.

Theorem 2: Let a TREKS key have length k. Then the key inference cost is $O(2k)$, making the computational cost of TREKS message despreading at most twice of conventional SS.

Proof: To prove the first part, we note that for each key bit, the receiver guesses two values and chooses the one yielding the highest number of peaks (see Figure 8.) This gives an inference cost of $O(2k)$. Because each key bit guess results in a despreading operation, every bit of the message is despread twice. The EoM cost is common for TREKS and conventional SS, proving the former to be at most twice as slow. Note that this cost can be reduced by discarding one of the two keys after attempting to despread a few bits of a packet segment. □

Given its small computational overhead, TREKS can be used in two ways. First, it may be used to share the secret required by conventional SS, communicating the challenge of any mutual authentication and key establishment protocol as discussed in [1]. Once the secret is transmitted, all data is spread with conventional SS. On the other hand, TREKS may also be used for long-term communication, without communicating secrets. This is accomplished by spreading the message directly while letting the receiver infer the spreading key. To use TREKS in this mode only requires a few implementation optimizations which we discuss in the following section.

5 Performance Evaluation

In this section, we evaluate the performance of TREKS using MATLAB simulations and on our test-bed. First, we discuss the performance metrics of our evaluation, the simulation setup and the hardware/software specification of experimental runs. Then, we will present the types of jammers we consider for TREKS evaluation, and finally the performance results with and without jammers.

5.1 Implementation

To implement TREKS we use the Universal Software Radio Peripheral (USRP) [21], and 3 different NVIDIA graphic cards [22] to implement a sender and three receivers with different computational powers. On the software side, we employ GNU Radio as SDR [23] and the NVIDIA Compute Unified Device Architecture (CUDA) for GPU programming [24]. Our PN-generation is implemented using AES with 128-bit keys [19]. See Figure 10 for an overview of the main system components.

We carried out our experiments in a cabled setup for two reasons: to achieve BER/PLR graph as a function of varying JSR while being isolated from surrounding interference, and reproducibility. Three separate host computers with USRPs represent sender, receiver, jammer. The USRPs' are cabled using 50Ω RF-SMA cables with 30dB attenuation on the receiver side [21]. All USRP boards are placed inside shielded enclosures to avoid external noise. For each experimental run, the sender sends 1 million 1024-bit packets.

Due to space limitations, we omit the implementation details of our testbed, which can be found in our Technical Report [25].

5.2 Performance Metrics

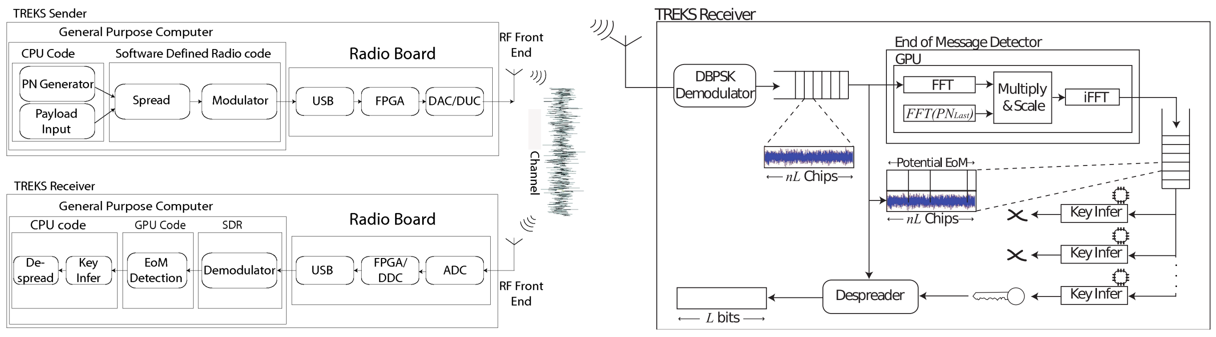
We evaluate the performance of TREKS in terms of decoding computation and storage costs, and Jamming resiliency. The jamming resiliency can be measured in terms of Bit Error Rate (BER), Packet Loss Rate (PLR), and computation delay sustained in the presence of jamming.

As we showed in Section 3, the computation cost in TREKS is at most twice that of conventional SS due to the key inferring process. Our evaluation will measure the average time it takes to complete a packet transmission from the sender to the receiver under different system specifications, and the contribution of each module to the total time.

In terms of storage cost, the receiver maintains a FIFO of size $2nl$ to buffer incoming signals and an additional buffer of size nl in GPU memory for the precomputed MAC address' FFT. This is a total of $3nl$ chips, which is clearly within the capacity of today's hardware.

We measure resiliency by BER and PLR. Sometimes, depending on the ability and availability of jamming resources, a jammer might want to simply increase the delay of message decoding instead of preventing the whole communication. To accomplish this, the jammer may insert partial or complete messages as described in Section 2. Such a strategy increases the number of False Positives (FPs) during EoM detection causing the decoding process to take longer than in conventional SS with pre-shared keys.

Note that the choice of the EoM Threshold directly affects the PLR/BER and FPs observed during the receiver's message decoding. We empirically choose an optimal value for the detection



(a) TREKS System (b) TREKS receiver processing blocks

Fig. 10: TREKS Architecture

threshold used in Figure 7. The details of this process are described in [25].

We consider three jamming strategies for evaluation: Gaussian, MAC, and random. The Gaussian jammer adds continuous AWGN into the channel. The MAC jammer inserts a random message spread with the PN sequence from the receiver's MAC address, targeting the last transmission bit. The random jammer behaves similar to the MAC jammer, except the seed is a random PN sequence.

In terms of energy efficiency we consider *Continuous* and *Non-continuous* jammers. Continuous jammers are non-budgeted and have resources to jam continuously. We implement Gaussian, MAC, and Random jammers for the continuous case. Non-continuous jammers, on the other hand, are budgeted and are forced to jam selectively. Since the adversary does not know the start of transmissions, non-continuous jammers have to memorylessly jam at some rate λ (a function of the jammer budget). We implement MAC and Random jammers for the non-continuous case. Further details about the choice of jammers can be found in [25].

To evaluate the non-continuous jammer, we consider a notion of discretized communication time with time slots of duration nl chips. The message transmission need not be synchronized with the beginning of a time slot. The jammer takes two parameters: λ and JSR. λ represents the probability of sending a jamming signal at a given time slot (this corresponds to discretization of a Poisson memoryless jammer to a Bernoulli jammer), and JSR is the jammer-to-signal power ratio. The cost to the jammer is $\lambda \times JSR$, and its goal is to maximize the PLR and BER for a given budget. Note that because the adversary does not know when transmissions happen, the cost of the jammer should be further scaled by a factor μ, representing the packet arrival rate. We consider the best case scenario for the jammer where $\mu = 1$ as well as the case where the jammer sends complete messages. A jammer may send partial messages but this can be independently addressed with appropriate interleaving and coding [5].

Parameter	**Value**
Spreading Factor, n	100
Packet Size, k	1024 bits
Key Size, n	10
Jammer to Signal Ratio, JSR	[-30dB . . . 20dB]
Frequency	2.4GHz
Modulation	Differential BPSK
Rate	1 Megachips per second
Normalized Signal Power	0 dBW
Noise Power	-20 dBW

TABLE 3: Simulation and Experiment parameters

Component	**Type**	**Model/Version**
Host OS	Ubuntu	v. 9.10
Host CPUs	Intel	Core2 Q9300
Receiver GPU	nVidia	GTX280, 9800GT, 8600GT
GPGPU Platform	CUDA	v. 2.2
SDR	GNU Radio	v. 3.2
Radio Board	USRP	v. 1

TABLE 4: Experimental Test-bed Specifications

We ran across two major problems while carrying out experiments on GNU Radio/USRP platform:

- *Saturation Effect:* Compared to the expected range for varying the amplitude of a sender signal according to the GNU Radio API docu-

Processing time	**Recvr 1**	**Recvr 2**	**Recvr 3**
GPU Model	GTX280	9800GT	8600GT
CPU (Core2)	Q9300	E8400	Q9300
EoM Detect	0.891	0.943	1.37
Key Infer	3.2	3.9	3.11
Despreading	1.6	1.95	1.55
GPU Specs			
Global Memory	1GB	512MB	512MB
Shared Memory	16KB	16KB	16KB
Registers/SM	16384	8192	8192
SMs	30	14	4
SPs	240	112	32
Retail Price	$250	$110	$40

TABLE 5: Average processing time (ms) of 1Kib packet, 20dB gain

mentation, we found out that the range without the sender reaching its saturation point is much smaller. The ADC ran at almost a full scale (saturated) by the time the range was half-way to the API provided range.

- *Granularity:* The mapping between the actual signal power (in dBm) and amplitude setting in the GNU Radio application is not precise. The power variation is deterministic only when the step-size is $1/10th$ of the allowed range.

We resolved these issues by attenuating the level of Jammer to Signal power (JSR) to stay within a range where USRPs behave correctly. However, at high JSR (BER under 10^{-6}) other factors dominate the performance and accuracy of the USRP boards.

5.3 Performance Results

We first look at the TREKS computation cost under our TREKS implementation. All plots are based on $10,000$ runs for each possible set of parameters (Table 3.) Due to limited space, we present graphs that correspond to packet size, data rate, spreading factor and encoding typical in wireless DSSS and feasible under the limitations of our hardware (see Table 4 for hardware and software specifications.)

5.3.1 Computation Cost

Table 5 shows the time of each task, and the hardware specifications of each GPU Model [24].

In order to sustain a 100M chip per second rate, the time for each element in the pipeline must take less than 1ms. The EoM Detection module falls below this mark due to its GPU implementation. The inferring and despreading modules account for most of the computation time, yet they can match

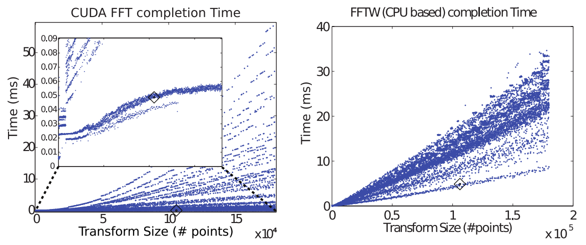


Fig. 11: Speed of CPU FFTW and GPU FFT.

the rate of the EoM Detector if the task is spread among CPU cores.

False positives may affect the performance of our receiver, but if key inferring for FPs fails in early rounds, resources are freed faster for new data blocks. Section 5.3.2 analyzes the FP rate in our system. In turn, the despreader will also benefit from having extra cores for instruction execution.

In terms of economic cost-benefit, the mid-range 9800GT performs better than the GTX280. Both of these GPUs can be used to obtain the 100Mchip rate, but the GTX offers only 5.5% improvement for twice the cost. Even if the 8600GT falls short of such mark, it is still suitable for rates of over 70Mcps.

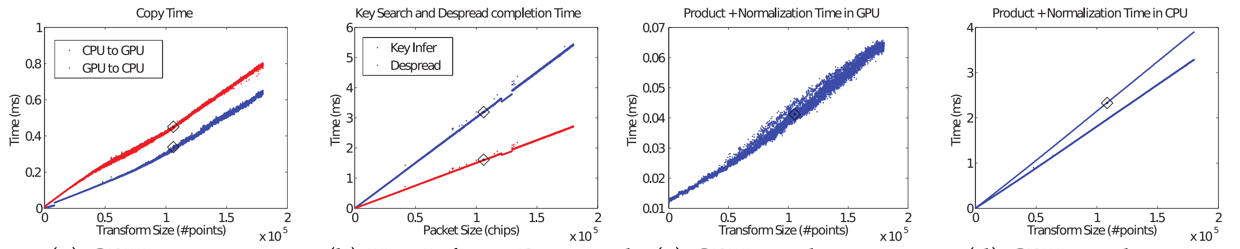
To analyze the performance of our EoM Detector module, we compare it against an EoM module built using FFTW3, a CPU-based FFT library [26]. We perform 100 runs of each with transform sizes ranging from 16 to 180000 points in 16-point steps.

The FFT speed depends on transform size (See Figure 11). Inputs that are powers of small prime numbers give the best performance on both cases, but the CUDA FFT implementation is clearly faster.

The product of transforms on running on the GPU gets 2 orders of magnitude improvement over a CPU implementation (Figure 12) because the product of individual points is performed in parallel. The plot shows that for every vector size, the GPU implementation is always faster even if it is slightly more scattered.

Transfer of data points to the GPU is a step required by the GPU-accelerated EoM Detector (see [25] for a detailed explanation) and does not exist in a CPU-based FFT implementation. Figure 12 shows the average time needed to complete copying in each direction. The GPU to CPU copy is less common in graphics operations, and thus is less optimized in hardware [24].

Full Key inferring and despreading times on one CPU core as shown by Figure 12 show that



(a) GPU copy time (b) Key Infer & Despread (c) GPU product time (d) CPU product time

Fig. 12: Receiver tasks speed. Diamond marks testbed parameter.

Model	In Copy	FFT	Prod.	iFFT	Out Copy
GTX280	325	40.3	43.1	39.0	443
9800GT	287	49.1	67.2	45.4	494
8600GT	478	42.9	239	40.8	567

TABLE 6: Times in μs for a 102400-size transform.

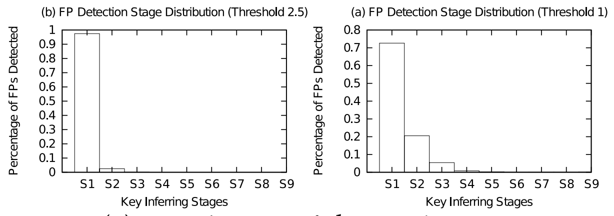
the TREKS receiver bottleneck resides on these two operations when using the accelerated EoM block. Since our design distributes each key search to every CPU core available, the time in average reduces to less than a millisecond with 4 cores (See Figure 10.)

An obvious optimization would have the Key Inferring and Despreading modules run on the GPU to benefit from the clear time savings, or even push every operation to the GPU. Not every algorithm translates well to a massively parallel device, however. For instance, the final addition operation when computing correlation acts as a barrier for parallelization, and the variable that contains it must be updated atomically, one thread per multiprocessor, defeating the purpose of the GPU. Thus, it is not clear a GPU implementation would always yield the savings seen on an FFT operation. As seen in Figure 11, even GPU implementations have border cases that run *slower* than in a CPU implementation.

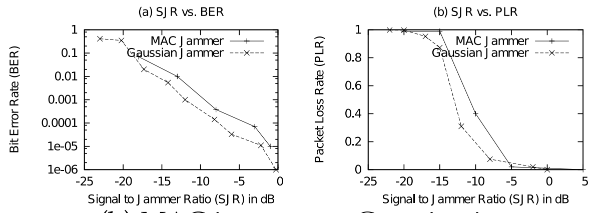
5.3.2 *Jamming Resiliency*

Now, we look at TREKS performance in terms of jamming resiliency against various types of jammers described above:

Figure 14 shows the PLR and the BER under TREKS in the presence of Gaussian jammers as a function of JSR. Note the imperfect gain of only about 15dB (instead of 20dB due to the spreading factor of $n = 100$) in TREKS BER and PLR graphs. This is mainly due to the imperfect synchronization, the empirical choice of threshold, and the



(a) Distribution of the FP detection stage



(b) MAC jammer vs. Gaussian jammer

Fig. 13: MAC and Gaussian Jammer Performance

USRP/GNU Radio hardware limitations.

In terms of FPs, Figure 13 shows that most of the FPs are detected by the first two stages of key inferring, for both simulation and experimentation. Hence, FPs do not impact TREKS's computation by much when compared to the decoding.

Figure 13 shows the performance of TREKS against the MAC jammer. There is little difference in performance because a MAC jammer in general only increases the FP rate. To destroy the EoM, the jammer must be synchronized at the chip level which is highly unlikely, showing that TREKS is resilient against even the most effective of the jammer types.

For the evaluation of a non-continuous, memoryless jammer with fixed rate λ, we consider a time-slotted communication model (See Figure 15). Consider a sender message spanning two consecutive time slots. Then, there are four possible scenarios: (1) Only the first TS is jammed, (2) Only the second

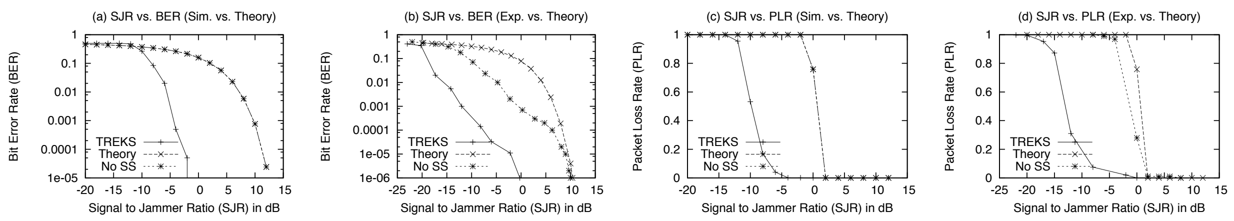


Fig. 14: Performance of TREKS against gaussian jammer (Simulation and Experimentation)

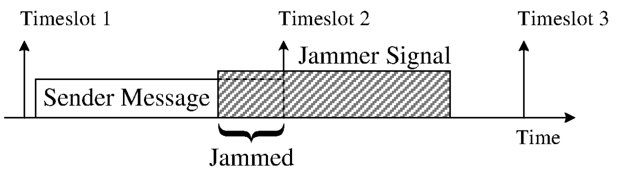


Fig. 15: Timeslotted Communication Time

TS is jammed, (3) Both TS are jammed, and (4) None of the TSs are jammed. Scenario (1) only impacts key inferring (BER/PLR), Scenario (2) impacts EoM detection (FP rate). Scenario (3) impacts both processes as shown by Figure 16. Obviously Scenario-4 does not have any impact in TREKS performance.

Similar to the continuous jamming case, we find that there is hardly any difference between the impact (PLR, BER, FP) of the MAC jammer and the Random jammer against TREKS. Now, given all possible scenarios and a fixed λ (jamming rate), we can calculate the expected PLR for a non-continuous jammer as follows:

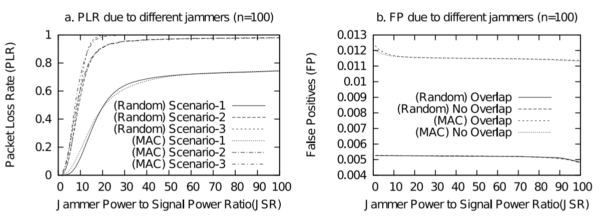
$$E[PLR] = E_1\lambda(1-\lambda) + E_2\lambda(1-\lambda) + E_3\lambda^2 + E_4(1-\lambda)^2$$

where E_1, E_2, E_3,and E_4 are the expected PLR for the above Scenarios, respectively. Figure 16 shows the $E[PLR]$ as a function of a given budget. We observe that the MAC and the random jammers attain their optimum approximately when $10 \leq JSR \leq 15$. This implies, even in the best case scenario for the jammer when $\mu = 1$, the jammer needs to spend 10 times more energy to reduce the throughput to 30%. For $\mu = 0.1$, the jammer would have to spend 100 times more energy to have the same effect.

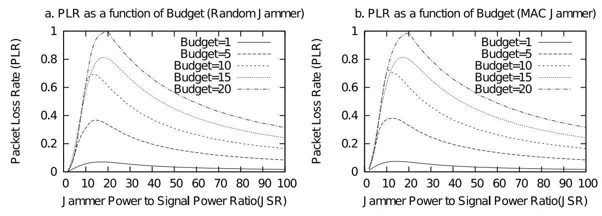
Lastly, any jammer strategy that does not use the destination MAC address as the seed for its spreading sequence, its signal is reduced to noise under TREKS with a factor or n (see Theorem 2.)

6 Conclusion

In this paper we propose new mechanisms, design and a full implementation of a real-time direct sequence spread spectrum system that does not require pre-shared secrets between the parties. We use readily available components to build our demonstrator, displaying four orders of magnitude improvement of computation cost in comparison to existing schemes. We are able to sustain (in terms of computation) a 1Mbps bit-rate spread by a factor of a 100 (i.e., 100 mega-chips per second) spread over 200Mhz bandwidth. Finally, we evaluate both the computation cost and the achieved resiliency.



(a) Function of Varying JSR



(b) Under fixed budget

Fig. 16: Comparison of Jammer Performance

References

[1] T. Jin, G. Noubir, and B. Thapa, "Zero pre-shared secret key establishment in the presence of jammers," in *MobiHoc '09*.

[2] M. K. Simon, J. K. Omura, R. A. Scholtz, and B. K. Levitt, *Spread spectrum communications; vols. 1-3*. NY: Computer Science Press, Inc., 1986.

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

[4] 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.

[5] G. Lin and G. Noubir, "On link layer denial of service in data wireless lans," *Wirel. Commun. Mob. Comput.*, vol. 5, no. 3, pp. 273–284, 2005.

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

[7] M. A. Bender, M. Farach-Colton, S. He, B. C. Kuszmaul, and C. E. Leiserson, "Adversarial contention resolution for simple channels," in *SPAA*, 2005.

[8] B. Awerbuch, A. Richa, and C. Scheideler, "A jamming-resistant mac protocol for single-hop wireless networks," in *ACM PODC*, 2008.

[9] W. Xu, K. Ma, W. Trappe, and Y. Zhang, "Jamming sensor networks: attack and defense strategies," *IEEE Network*, vol. 20, no. 3, pp. 41–47, 2006.

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

[11] J. Chiang and Y.-C. Hu, "Cross-layer jamming detection and mitigation in wireless broadcast networks," in *MobiCom '07*.

[12] A. Chan, X. Liu, G. Noubir, and B. Thapa, "Control channel jamming: Resilience and identification of traitors," in *ISIT '07*.

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

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

[15] 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.

[16] M. Strasser, C. Popper, and S. Capkun, "Efficient uncoordinated fhss anti-jamming communication," in *MobiHoc*, 2009.

[17] D. Slater, P. Tague, R. Poovendran, and B. Matt, "A coding-theoretic approach for efficient message verification over insecure channels," in *2nd ACM Conference on Wireless Network Security (WiSec)*, 2009.

[18] C. Popper, M. Strasser, and S. Capkun, "Jamming-resistant broadcast communication without shared keys," in *USENIX Security Symposium*, 2009.

[19] J. Daemen and V. Rijmen, "The Rijndael Block Cipher," 2003. [Online]. Available: http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf

[20] A. Bogdanov, D. Khovratovich, and C. Rechberger, "Biclique Cryptanalysis of the Full AES," 2011. [Online]. Available: http://research.microsoft.com/en-us/projects/cryptanalysis/aesbc.pdf

[21] Ettus Research, "Universal software radio peripheral." [Online]. Available: https://www.ettus.com

[22] NVIDIA, "Compute unified device architecture." [Online]. Available: http://developer.nvidia.com/object/cuda.html

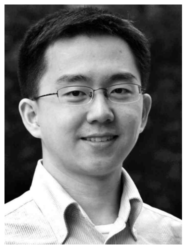
[23] GNU Radio Project, "GNU Radio." [Online]. Available: https://gnuradio.org

[24] NVIDIA, "Compute Unified Device Architecture Programming Guide v.2.2," 2009. [Online]. Available: http://www.nvidia.com/object/cuda_develop.html

[25] A. Cassola, T. Jin, G. Noubir, and B. Thapa, "Spread spectrum communication without any pre-shared secret," http://www.ccs.neu.edu/home/bthapa/techreport/trek.pdf, Tech. Rep., 2010.

[26] M. Frigo and S. Johnson, "The design and implementation of fftw3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, Feb. 2005.

Aldo Cassola received his BS from the Polytechnic School at San Francisco University, Quito in 2001, and his MS at CCIS, Northeastern University, Boston in 2008. Currently he is working towards his PhD in Computer Science at CCIS, Northeastern University. His work focuses on the study and implementation of multi-layer secure wireless networking, distributed systems, and parallel computing.



Tao Jin received his BS degree in Computer Science from Peking University in 2005. He is currently a PhD student in the College of Computer and Information Science at Northeastern University. He worked at Nokia Research Lab, Palo Alto as Research Intern in 2007 and 2008. His present research interests include mobile and ubiquitous computing, wireless networks, and distributed systems.

Guevara Noubir 's research covers both theoretical and practical aspects of secure and robust wireless communication systems. He holds a PhD in Computer Science from the Swiss Federal Institute of Technology in Lausanne (1996). He is a Professor of Computer Science at Northeastern University since 2001. He was a senior research scientist at CSEM SA (Switzerland) between 1997 and 2000 where he led several research projects and contributed to the definition of the third generation Universal Mobile Telecommunication System (UMTS). He is a recipient of the NSF CAREER Award. Dr. Noubir held visiting positions at Eurecom, MIT, and UNL. He is a Senior Member of the IEEE, and a member of the ACM.

Bishal Thapa earned his B.A/M.A in Mathematics from State University of New York in May 2005, and his MS in 2007 and PhD in Jan 2011 in Computer Science from Northeastern University, Boston. His PhD work focused on developing Robust Wireless Communication Systems in the presence of Adversaries. Currently, he works as a Research Scientist at Raytheon BBN Technology. One of his recent DARPA projects includes developing a wireless communication protocol for Fractionated Satellite System set to be launched in early 2015.