Zero Pre-shared Secret

Key Establishment in the presence of Jammers

-Tao Jin, Guevara Noubir, **Bishal Thapa**

Motivation

- Wireless Communication vulnerable to attacks
  - Broadcast medium is open to adversarial attacks
  - Resiliency against malicious behavior is of great significance

- Spread Spectrum (SS)
  - One of the most efficient anti-jamming mechanisms
  - Mostly used in military applications, but gaining civilian interest
  - Biggest Limitation:
    - Requires pre-shared key, thus usually not considered fit for usage in:
      - Systems that employ large number of associating/disassociating nodes
      - Lack of a safe medium to pre-share secret in the presence of jammers
    - Problem described as the *Circular Dependency Problem [CDP]*.

CDP Formulation and Previous Work

- M. Strasser, C. Popper, S. Capkun, and M. Cagalj. *Jamming-resistant key establishment using uncoordinated frequency hopping*. In ISSP, 2008
  - First to formulate CDP
  - Proposed Uncoordinated Frequency Hopping [UFH] as a solution
    - New SS anti-jamming technique that does not rely on shared secret
      - Basic idea of UFH:
        - Sender randomly picks frequency channels to hop on w/o a coordination
        - After sufficient transmissions, sender/receiver settle on the same channel
      - Establishes key that can be used to support coordinated FH (CFH)
    - Achieves similar jamming resiliency as that of CFH
  - Lower communication throughput and incurs higher storage/processing costs due to transmission repetitions.

Our Contribution

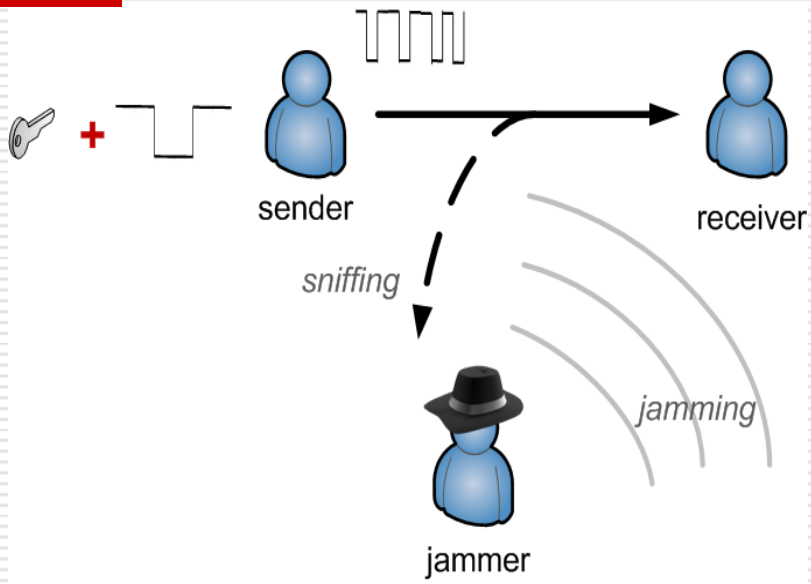
- Propose a communication paradigm
  - Intractable forward-decoding
  - Efficient backward-decoding
- Based on such a paradigm, we propose **TREKS**
  - **T**ime-**R**eversed **M**essage **E**xtraction and **K**ey **S**cheduling
    - Zero pre-shared key
    - Efficient anti-jamming technique that can be used to establish a SS key'
    - Zero energy overhead compared to the traditional SS
    - Propose optimized decoding mechanism
    - Essentially low communication overhead, computation, and storage cost
  - Show that the computational cost of **TREKS** is at most twice the computational cost of traditional SS system with a pre-shared key.

Outline

- System Model and Adversary Model
- Time-Reversed Key Scheduling in DSSS
  - Na ïve ZPK-DSSS
  - ZPK-DSSS with key scheduling
- Message Extraction
  - Phase I: End of Message (EoM) Detection
    - Algorithm and Design
  - Phase II: Spreading-Key Infer and Message Despreading
    - Algorithm and Design
- Performance Evaluation
- Conclusion and Future Work

System Model

- Systems that are traditionally capable of doing SS
- Assumptions:
  - CA signed public key
  - Protocol publically known to sender, receiver and jammer



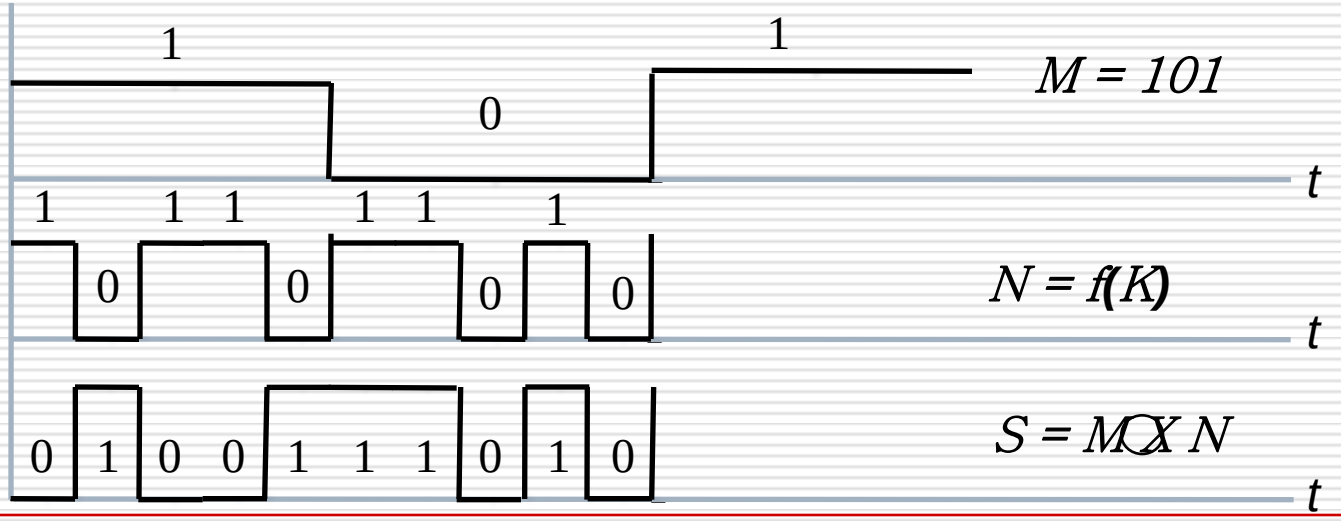
- Our work is focused on allowing a secure transmission of the information required for the key establishment process over an unsafe medium

Adversary Model

- Co-located with the sender and the receiver
- Jammer's primary goal:
  - Prevent the successful reception of sender's message (Packet Loss)
  - Make the original message undecodable (High Bit Error)
  - Increase the delay of message extraction (Denial of Service Attack)
  - Increase computation and storage cost (Message Extraction Failure)
  - Damage the integrity of the message
- Assumptions:
  - Ignore the gain of configuring physical layer parameters
  - Adversary cannot tunnel the signals for brute-forcing before the end of the message transmission (few milliseconds)

Zero Pre-shared Key DSSS

- Notations:
  - *M :* Plain message, $|M| = l$ *bits*
  - *K :* Spreading Key, $|K| = k$ *bits*
  - *N :* Spread Sequence, $N = f(K)$ and $|N| = n$ *bits*
  - *S :* Spread Message, $|S| = n \times l$ *bits*

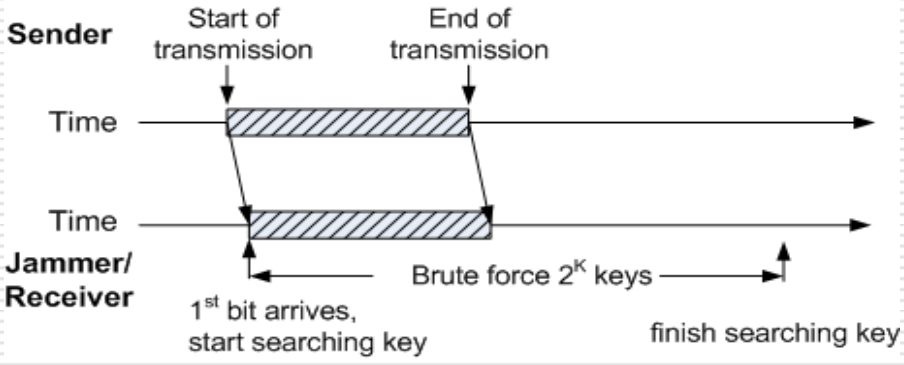


Zero Pre-shared Key DSSS

- In ZPK-DSSS, to despread the message it requires a
  - Brute-force search **$O(2^k)$ keys**
    - No knowledge of the communication happening
    - The search has to occur per incoming chip
- Thus, jamming is inefficient
  - Needs to know the spreading-key *K* in order to jam efficiently
  - Jammer Resilient
    - Impossible if *k is chosen* to be large enough
  - Applies to **both** the receiver and the jammer (Limitation)

I. Intractable Forward Decoding

- Message **delivered** before the key is broken!



- Security definition**:**
  - We call the key size ***k* secure** if

    given, $T_{trans}(l)$ - Transmission time of l bits

    $T_s(k)$ - Time required to brute-force search a k-bit key

$$T_{trans}(l) << T_s(k)$$

I. Intractable Forward Decoding (Contd..)

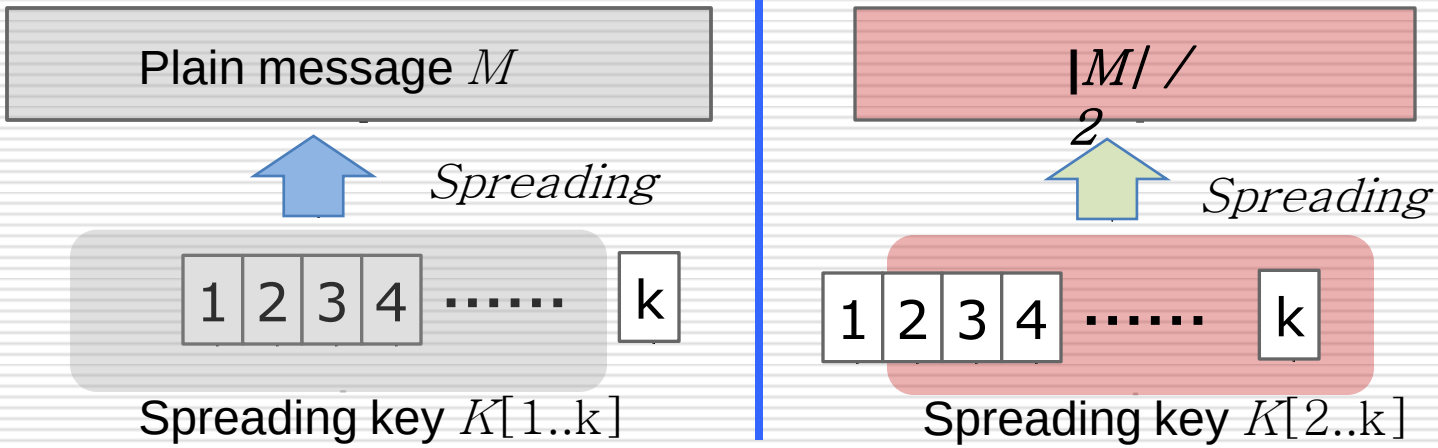
- Resiliency and Limitation go hand-in-hand
  - Jamming Resiliency
    - Basically, chips are collected first then processed
      - Luxury that jammers do not have
      - Late for jamming, but *OK* for key establishment
  - Limitation
    - Resiliency feature is also the limitation
      - High computation overhead for the receiver to process the samples
      - Impractical for real-time communication, $O(2^k)$ per incoming chip
- Exploit the luxury of the receiver to be able to process after the communication is over
  - Efficient Backward Decoding

II. Efficient Backward-Decoding

- Idea: Sender weakens the spreading-key as communication nears the end while ***maintaining the key-security***
- Maintaining the key security
  - The message is divided into segments from start to the end
  - A *weaker* key (easier to guess) is used to spread the later segments
    - Weaker: Less information is revealed about the key
    - Near the end, the key used to spread the last segment might have just one-bit entropy
  - However, the security is still maintained, i.e.,
    - $T_{trans}(l) << T_s(k)$ if l is the last segment and k is the key used spread the last segment
    - Done by implementing a ***key schedule*** into the spreading

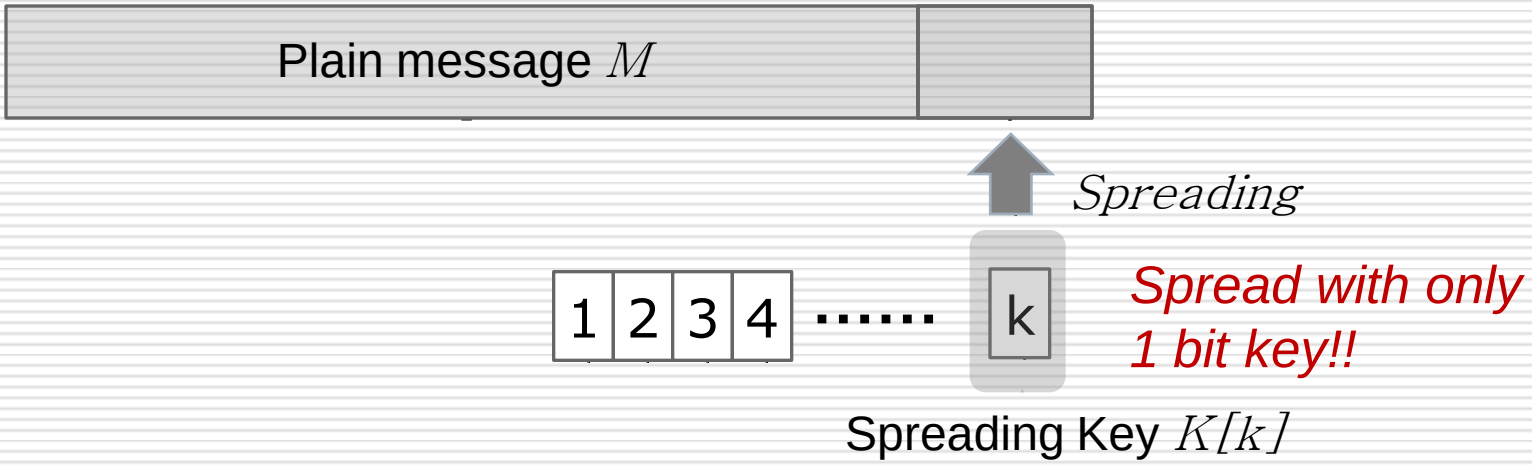
II. Efficient Backward-Decoding (Contd..)

- ZPK-DSSS Spreading:
  - Each message bit spread with a *n*-bit PN-sequence generated by having a *k*-bit key as a seed
  - From here onwards, we use the PN-sequence and the key interchangeably
- With a Key Schedule:



II. Efficient Backward-Decoding (Contd..)

- Key schedule used in TREKS:



$$T_{trans}(l) \ll T_s(k) \quad \Rightarrow \quad T_{trans}\left(\frac{|M|}{2^i}\right) \ll T_s(k-i)$$

II. Efficient Backward Decoding (Contd..)

- Result:
  - ***Efficient*** backward decoding:
    - Only two possibilities for the last key used to spread the last segment
    - By induction, we see that inferring the spreading key backward in time with two possibilities at each key bit (time-reversed decoding)
      - Cost is reduced from *O(2^k) to O(2k)*

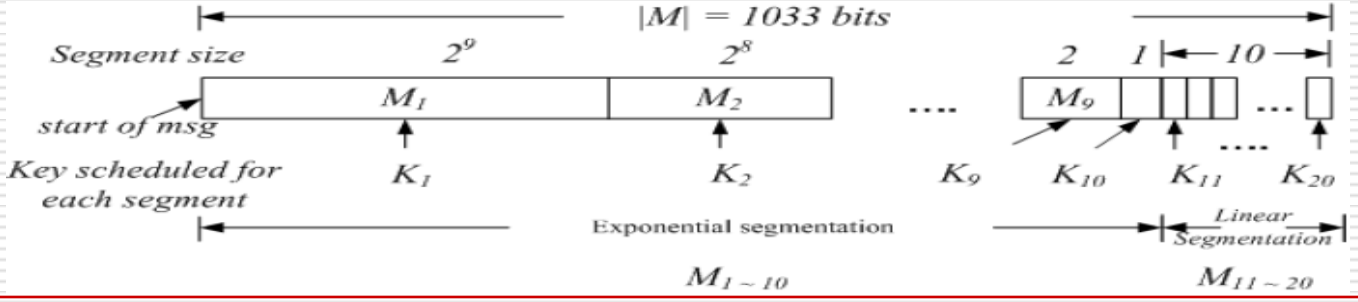
- Theorem: The computation cost of **TREKS** message despreading is at most twice the cost of message extraction in a traditional SS system with a pre-shared secret.

Optimized Decoding

Observation 1:

Message size has to be too long to have the key scheduled as above

- For e.g.: *k=20* will require a message size of length at least 1Mbits.
- **Optimization 1 -** *Key Scheduling w/ Linear Tail*
  - *Linear Tail:* For last *x* bits, we allow the sender to weaken the key size at a linear rate of 1 key-bit per message-bit.
    - *x* is picked in such a way that $T_{trans}(x\ bits) < T_{\delta}$, where $T_{trans}(x\ bits)$ is the transmission time for *x* bits over the air and T_{δ} is the radio turn-around time
    - Though this implies it is equally easier for the adversary to figure out the key used to spread the last *x* bits as it is for the receiver, $T_{trans}(x\ bits)$ is too small to jam



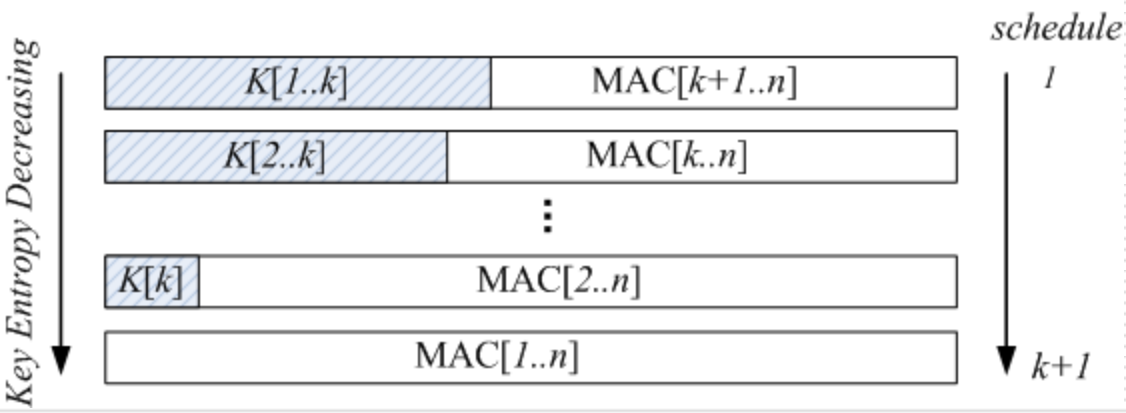
Optimized Decoding (Contd..)

Observation 2:

The key used to spread the last segment is *overly exposed*

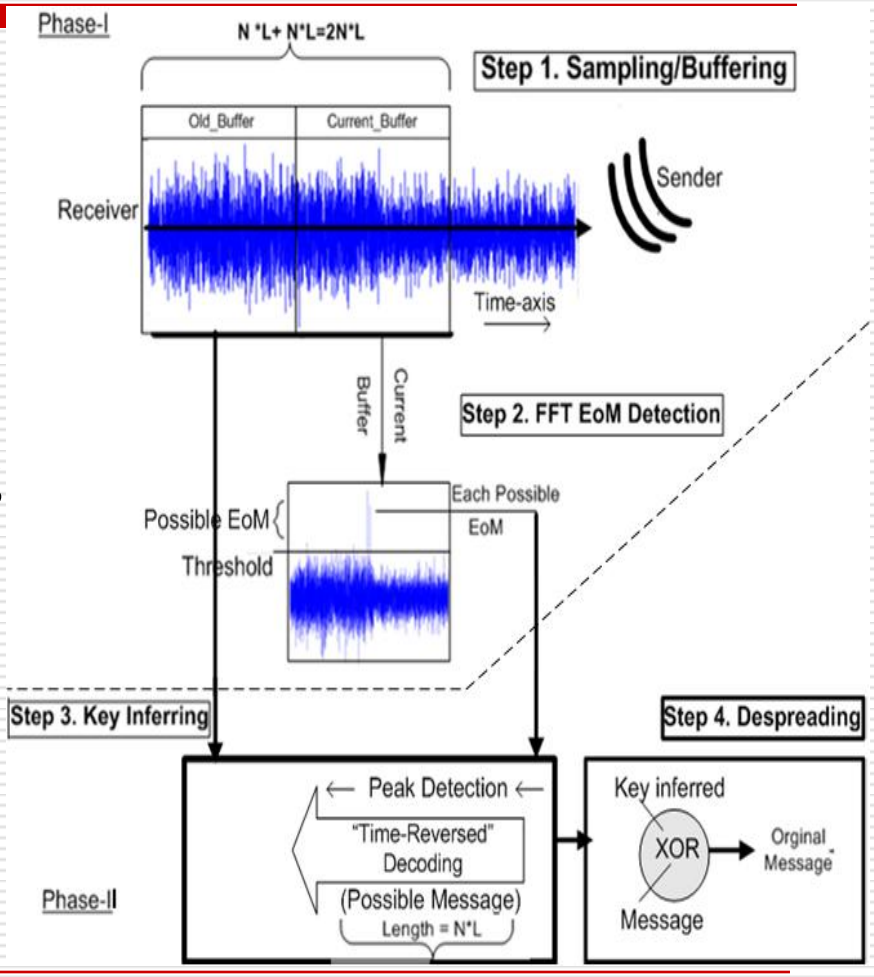
- The bit entropy of the last key is only one bit
- Jammer can continuously jam the last message segment affecting all receivers
- **Optimization 2 –** *MAC-masked key schedule*
  - Each key schedule is masked with a part of the receiver's MAC address

Potential 0/1 attack becomes a destination-oriented attack

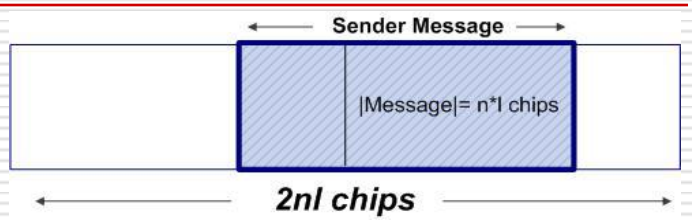


Message Extraction

- A two-phase process:
  - Reverse-time decoding
    - Phase I: Finding the EoM
      - Sampling/Buffering
      - EoM Detection using FFT
    - Phase II: Message Decoding
      - Backward Inferring of the Spreading-Key
      - Despreading the Message
      - Message Integrity Verification



Finding the EoM (Phase-I)



- Phase-I a: Sampling/Buffering
  - Signal Samples buffered into a FIFO
  - Only 2*n*l chips is kept in the buffer at anytime
    - Whole message of length n*l will be inside the block
- Phase-I b: EoM detection using FFT
  - Bit synchronization is achieved by computing the cross correlation between the received signal and PN-sequence candidates
    - Use FFT to reduce the computation cost from $O(N^2)$ to O(NlogN)
    - *Threshold* is picked empirically (explained later in the eval section)
  - Optimization
    - Unlike the traditional SS, in TREKS, we process a batch of n*l chips instead of n-chips (a bit) at a time

EoM Detection using FFT

Animated matlab plots to show the correlation vector and the peak detection

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

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;}
```

Despreading the Message (Phase-II)

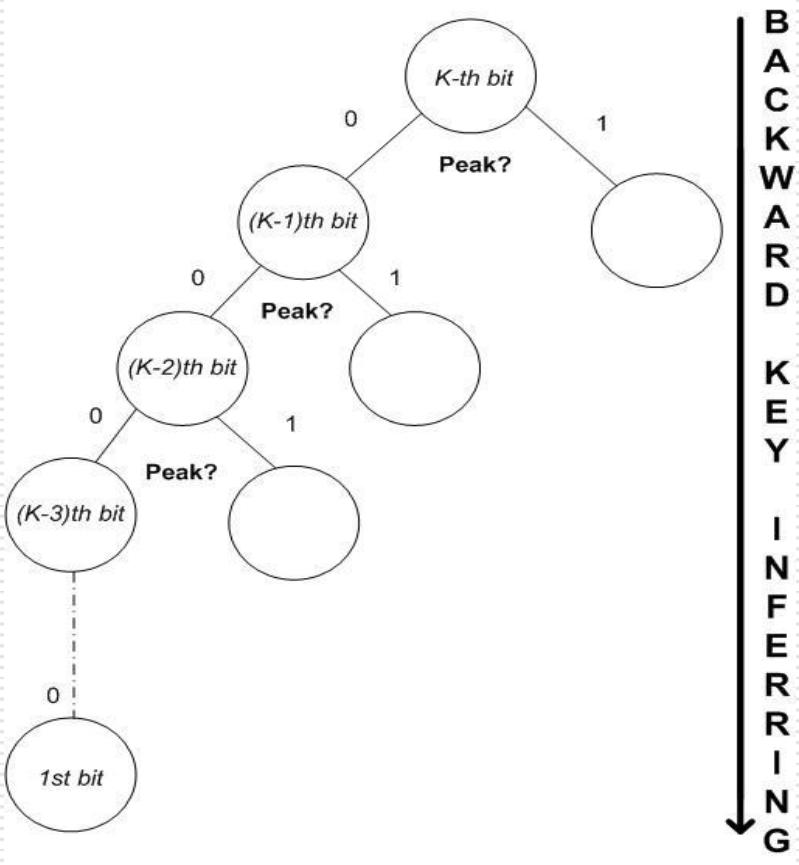


Fig: Reveal a key-bit at each stage of inferring

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;
}
```

Despreading the Message (Phase-II Contd..)

- Phase-II a: Inferring the spreading-key
  - Once a candidate for EoM is identified
    - Do backward decoding (two guesses at each key-bit position)
  - Multiple candidates for EoM is possible (*False Positives*)
    - Computation delay, but negligible compared to overall decoding cost
  - At each segment, 50% of the expected bits detection imply a true positive
- Phase-II b: Once all key-bits are inferred, message despread
- Phase-II c: Message Integrity verification
  - 160-bit Elliptic Curve based Digital Signature Algorithm (ECDSA)
- Key Establishment Protocol
  - ECDH, a possible choice

Performance Evaluation

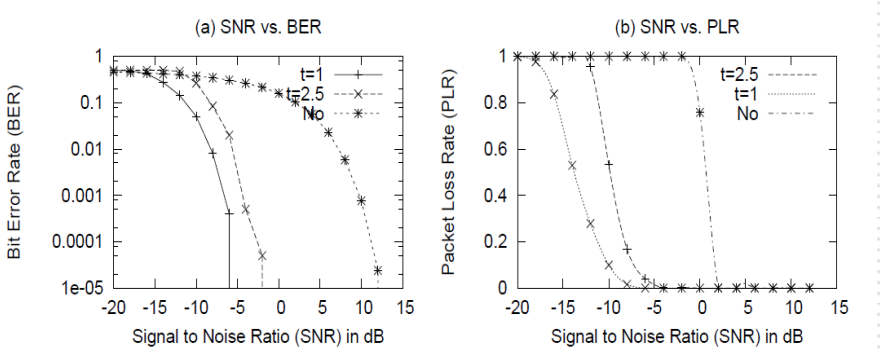
- Matlab Simulation
- Simulation Parameters:
  - Spreading Factor (n) = 100
  - Packet Size (l) = 1033 bits
  - Key Size (k) = 19
  - Jammer Power to Signal Power Ratio (JSR) = [1…100]
  - Normalized Signal Power = 0 dBW
  - Noise Power = -20 dBW
- Evaluation Metric:
  - Packet Loss Rate (PLR)
  - False Positives (FP)
  - Computation and Storage Cost

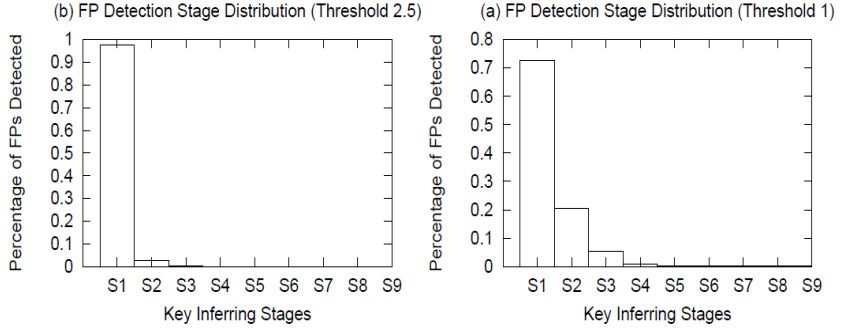
Performance Evaluation (Contd..)

- Evaluation Categories:
  - Based on FPs generated during the message extraction process and PLR, we generalize our adversary model to the following:
    - TREKS vs. Gaussian Jammers (Noise-Only)
    - TREKS vs. **λ**-jammer
      - **λ**: *The probability that a jammer sends a jamming message at a given time-slot under an assumption that the communication time is discretized into timeslots of duration n*l chips.*
      - We also assume that the sender is always sending the message
        - Note: This benefits the jammer under our model. In reality, it is less effective.
      - Two sub-kinds of **λ**-jammers:
        - Random Jammer : Inserts an l-bit message, each bit spread with a random PN-Seq
        - MAC Jammer: Inserts an l-bit message, each bit spread with a PN-Seq generated by using MAC address of the receiver as seed.

TREKS vs. Gaussian Jammer

- PLR
  - PLR vs. SNR
    - Varying threshold
      - *t = 2.5 vs. t = 1*
      - *No noise (ideal)*
- BER
  - SNR vs. BER
    - Varying threshold
      - *t = 2.5 vs. t = 1*
      - *No noise (ideal)*
- FP
  - Only first two stages





Picking the Right *Threshold*

- *Threshold Value*
  - Determines the peaks of the correlation vector used for bit-detection.
  - Under our model, threshold = *t*avg* where *avg* is the average of the elements in the correlation vector.
  - We empirically decide the *threshold* value to be **2.5** based on PLR and FP it produces when evaluated against the scenario with noise-only.

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%

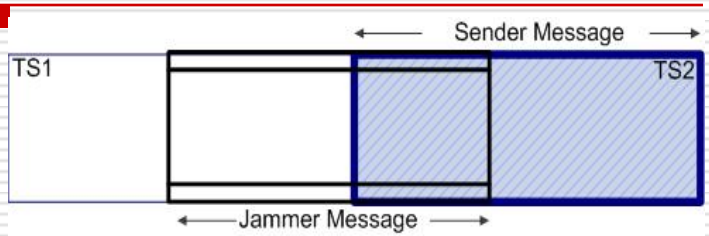
⇐ Packet Loss Rate

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%

⇐ False Positives

TREKS vs. λ-Jammer

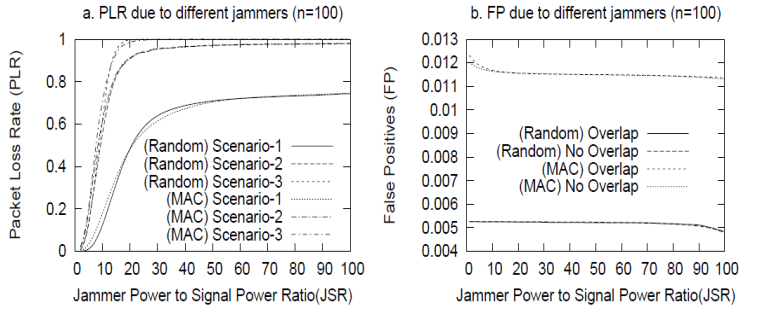
- Jamming Scenarios:



  - Based on the occurrence of the jammer message in a 2-TS block
    - Scenario-1: Jammer message occurs in the first TS.
      - Impact: Key-inferring process.
    - Scenario-2: Jammer message occurs in the second TS.
      - Impact: EoM detection.
    - Scenario-3: Jammer message intersects both TS.
      - Impact: Key-inferring and EoM detection.
    - Scenario-4: Jammer message misses both TS.
      - Impact: None.
    - Scenario-5: Jammer message perfectly synchronized with sender's.
      - Impact: EoM detection.

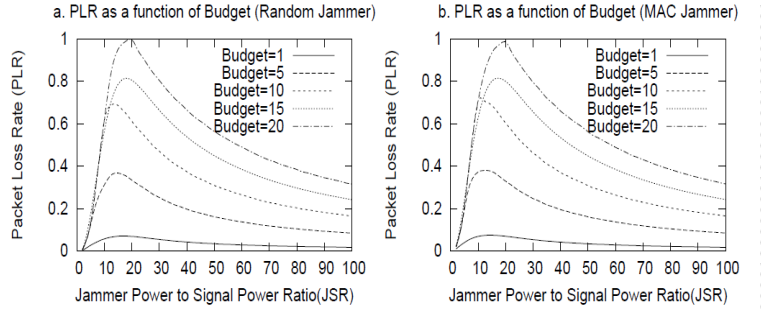
Simulation Results

- PLR vs. JSR
  - Scenarios: Scenario-3
  - Jammer Type : MAC
- FP vs. JSR
  - Scenarios: Indifferent
  - Jammer Type: MAC
- Expected PLR vs. Budget
  - Optimum budget
    - MAC: When $10 \leq JSR \leq 15$
    - Random: When $10 \leq JSR \leq 15$



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

Budget = λ * JSR



Computation Benchmark & Storage Cost

- Computation Benchmark:

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

- Storage Cost:
  - FIFO buffer size = (n*l)/4 bytes
  - Total number of messages recovered
    - If the jammer injects *j* messages, then we store at most ((j+1)*l)/8 bytes
    - Total storage cost = 4*n*l + ((j+1)*l)/8 bytes
      - Assuming each sample is a 32-bit I/Q value
    - Negligible. Clearly within the storage capacity of today's computers.

Conclusion & Future Work

- Conclusion
  - Establishing a SS system against jamming w/o pre-shared secret
  - Zero energy overhead in comparison to traditional SS system
  - Introduce a communication paradigm; intractable forward decoding and efficient backward decoding
  - Optimized decoding
  - TREKS Computation cost $\leq$ 2 * traditional SS communication cost
  - Allowing destination-specific transmission and inability to detect packet transmission until last few bits are transmitted.
- Future Work
  - Extension of TREKS for long-lived communication
  - Implementation of TREKS in a real-world system.