SYMBOL LEVEL
SECURITY KEY GENERATION
IN WIRELESS NETWORK
A THESIS
SUBMITTED TO GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE
MASTER OF SCIENCE
BY
ZHUOZHOU FU
DR. SHAOEN WU - ADVISOR
BALL STATE UNIVERSITY
MUNCIE, INDIANA
MAY 2016
Acknowledgement

I would like to say thanks to my advisor, the committee chair, Dr. Shaoen Wu. His continually help means a lot to me. He helped me not only in my research areas, giving advices and instructions, but also helped me growing in daily life. Without his help this thesis would not be possible.

I would like to express thanks to my committee members, Dr. Jeff Zhang and Professor Dolores Zage, for their strong support to my graduation. I would also say thanks to the department chair Dr. Paul Buis and the Director of Graduate Program Dr. Samuel Hsieh who offered me a lot of help in completing my graduation requirements.

At last, I want to use this chance to say thanks to all of my friends who helped me during me hard time and shared happiness and sorrow with me.
## Contents

1 Introduction ............................................................................................................................. 1

2 Related work ........................................................................................................................... 7

2.1 Time domain exploration .............................................................................................. 7

2.2 Frequent domain exploration ........................................................................................ 8

3 Methodology ......................................................................................................................... 10

3.1 A brief summary .......................................................................................................... 10

3.2 Noise-free channel measurement ................................................................................ 10

3.3 The ODFM inspired model .......................................................................................... 11

3.4 Key generation algorithms .......................................................................................... 13

   3.4.1 Step1: Quantization ........................................................................................... 14

   3.4.2 Step2: XOR operation ....................................................................................... 14

   3.4.3 Step3: Bit Reconciliation .................................................................................. 15

   3.4.4 Step4: Key Generation ...................................................................................... 16

3.5 A case study of $\lambda$ choice ....................................................................................... 16

4 Performance evaluation ........................................................................................................ 18

4.1 Set up .......................................................................................................................... 18

4.2 White noise model ....................................................................................................... 19

   4.2.1 Noise difference 5%, SNR vary ........................................................................ 19

   4.2.2 Noise Difference 10%, SNR vary ..................................................................... 21

   4.2.3 A brief comparison of the two experiments ...................................................... 22

   4.2.4 SNR 25, noise difference vary between 5% -35%............................................ 22
4.3 Rayleigh noise model

4.3.1 Noise difference 5%, SNR vary

4.3.2 Noise difference 10%, SNR vary

4.3.3 A brief comparison between tow noise differences

4.3.4 SNR 25, noise difference vary between 5% -35%

4.4 Ricean noise model

4.4.1 SNR 25, noise vary between 5% -35%

4.4.2 Noise difference 5%, SNR vary

4.4.3 Noise difference 10%, SNR vary

4.4.4 A brief comparison of two noise difference

5 Conclusion


1 Introduction

Today’s world is highly depending on all types of wireless devices. With the emerging of Internet of Things (IoT) where there are trillions of smart sensors, our dependence on wireless communication will become even more. The success of IoT largely depends on the reliability and security of the interconnection and exchanging of information. Conventional wireless security needs the Public Key Infrastructure (PKI) to support secure communication. With the emergence of billions of smart devices and sensors, many of them can’t access the PKI easily in their peer-to-peer communications. Instead, many researchers have conducted investigations about the security for infrastructure-less peer-to-peer communications. Exploring the uniqueness of wireless channel can be a remedy for the solutions [1]. One of the novel solutions is to exploit the principle of channel reciprocity to generate encryption keys between two wireless nodes.

Figure 1: Classic Wireless Security Key Generation Model.
A classic model in wireless security key generation is shown on Figure 1. Alice and Bob are users that are talking to each other and Eve is the third person who is trying to peek the information between Alice and Bob. Channel reciprocity, as referred as channel mutuality, refers to the fact that the wireless channel between Alice and Bob is symmetric and mutual to each of them. Because of that principle, Alice and Bob should ideally see the same channel response $h(t)$ at its own side independently. According to wireless information-theoretic security, as long as Eve is a half of carrier wavelength away from Alice and Bob, Eve will not able to see the same channel response as Alice and Bob see. In other words, the channel response $h'(t)$ that Eve can see between her and Alice and $h''(t)$ between her and Bob will be different, at least to some degree, from $h(t)$ between Alice and Bob due to channel’s uncorrelated random variations over space, time and frequency [2] [3]. In practice, if we are talking about 2.4 GHz WLAN, a half carrier wavelength is as small as 0.125m and it would not possible for Eve to directly detect the channel reciprocity between Alice and Bob, because that distance is within the eyesight range for both. Due to this fact, we can safely use the characteristics of channel response between Alice and Bob to generate the shared secret keys independently at each side for encrypting wireless communication without revealing useful information to Eve.

Different solutions for generating security keys are mainly different in how to extract the channel response. Normally Alice and Bob first use Time Division Duplex (TDD) to exchange the probing frames between them. They then independently measure the received frames at their own sides to calculate the channel response. How effective a solution can generate security key highly depends on how the channel response can be obtained. Wireless signal
strength is widely used by literature solutions as a metric to obtain channel response, including signal to noise ratio (SNR) and received signal strength indicator (RSSI) [4] [5].

Even though channel response is symmetric, the received signal will not be exactly symmetric at a pair of nodes due to the influence of noise and likely interference. A received signal at time t can be denoted as \( r(t) = h(t) \cdot s(t) + n(t) \), where \( n(t) \) is the noise, \( s(t) \) is the signal being transmitted and \( h(t) \) is the channel response. [6] Because noise is not likely to be symmetric and, as a result for the same symbol transmitted between Alice and Bob, the \( r(t) \) at each side will be different. It is necessary to cancel the influence of the noise in order to get the actual channel response.

In order to get channel response, many traditional solutions will take the measurement of SNR or RSSI that come from frame header at the physical layer when a receiver gets the signal \( r(t) \). These solutions have some problems, which are critical in wireless information security.

The first problem is that the solutions will take long time to get enough bits for encryption since only one measurement is taken every probing frame, the following things might happen:

- Since there are many probing frames needed to generate a key of over a hundred bits by the key generation process, more battery power is used. This is unacceptable in wireless communication because battery is one of the biggest limitations for wireless devices. A significant amount of work has been done to make a good balance between energy consumption and key generation.

- Because the process will take a significant long time to generate security keys, the vulnerability of the generated key is enlarged. Longer time taken up by key generation means longer time for Eve to do hacking, which results in a potential
security issue.

The second problem of literature solutions is that some of them did not take the influence of noise into consideration and the measurements cannot be precise in terms of revealing real channel response. Because every probing frame is affected by the asymmetric noise, Alice and Bob cannot have the exact channel measurements. As a result, it will increase the chance of having high bit disagreement rate. Without eliminating the noise, it is not likely to have channel response measured with high accordance.

The third issue is that as long as the measurement is only performed on the header of a frame, the rest of the frame is wasted. In other words, we are transmitting many useless signals. If we could also use the rest of the frame to perform channel response measurements, more bits can be generated with relatively low battery consumption.

The solution we propose in this thesis is based on a previous work done by Song and Wu [6], we keep all the merits that the previous solution had proposed but endeavor to increase the key generation rate (KGR).

The overall solution is summarized as five steps: First we have Alice and Bob transmitting specially designed frames in TDD. Second, once they all receive the transmitted frame they will do calculations independently at each side to have a matrix representing the channel response and the matrix is then digitized into a binary matrix. Third, we apply exclusive-OR (XOR) operation on the bit matrix and have a queue of XORed bits and a XORed bit matrix individually at both sides. Fourth, we identify the central indices for \( \lambda \) consecutive bits of 1s or 0s, keep them in a list and send the list to each other. Fifth, with the list of indices generated and received at both Alice and Bob side we find the common parts and use the common bit
The solution proposed in this thesis has advantages over traditional solutions:

- The solution takes the channel response measurements at the symbol level, thus it is able to get more measurements done with the same number of frames to be transmitted.

- A more efficient approach is designed to extract channel response from the previous work [6], thus we can have more bits generated for security keys.

- The solution handles the received symbols in a special way that it is able to get rid of the influence of random noise. This practice will make our channel response measurements to be more accurate and higher KGR compared to other traditional solutions while the Bit Disagreement Rate (BDR) as low as in the previous work [6].

- No garbage bits is produced in the solution and it makes full use of the wireless channel resource. In other words, we are generating the security key in a more efficient way and more resource can be saved, including channel and battery.

Compared with the previous work [6], this paper makes the following contributions.

We use the received matrix exhaustively and have more bits generated for security keys than before. During the process of calculating channel response, we use a different way from the previous [6]’s and more channel responses measurements can be calculated. For 12 symbols being transmitted, our solution can have 9 channel response measurements but the previous work can only generate 3.

If increasing the number of bits to be generated is at the cost of reducing entropy, then the
solution might not be a good choice. Our solution can increase the number of bits to be generated for security keys while keeping the entropy as high as before. In fact, the entropy value of the security keys generated by our solution is even slightly higher than those in the previous work [6]’s.

Furthermore, with higher key generation rates, the BDR remains as low as the work done before. A low BDR is vital in using the generated security keys. Large BDR can cause security keys useless for encryption if they can’t be reconciled. For the security keys generated by our solution, the BDR is just slightly higher than [6]’s work under some conditions.

We propose a standard metric of measuring key generation effectiveness. Instead of measuring how many key bits can be generated per frame we use how many key bits can be generated per 100 bits transmitted. Because different solutions may have different frame sizes, a standard metric of evaluation is fair in terms of the performance comparison.

We have conducted the performance evaluation on the proposed solution with Matlab for three different noise models to extensively test its effectiveness. Under various settings, the solution always performances as well as the previous solution over BDR and entropy value. At the same time, our solution can triple the key generation bit rate.

The rest of this thesis is ordered as following. Related work is reviewed in section II. Section III presents the details of our solution, including the design as well as the algorithms related to generating security keys. In Section IV, we evaluate our solution with experiment data to demonstrate the effectiveness. Following that will be a conclusion to conclude the thesis.
2 Related work

Researchers have investigated for years in exporting the wireless channel properties to generate shared security keys between two trusted users [4], [5], [7]-[14]. Thanks to [15], the analysis in that article inspired many researchers to propose many works related with wireless security keys generation either theoretically or experimentally [1], [4], [5], [8], [10], [16], [17]. Some of the solutions collected data that represent the reciprocal channel response based on signal strength of time-varying channel with deep fading.

2.1 Time domain exploration

In [5]’s work they started to use the characters of the wireless channels to generate security keys in order to get rid of the constriction of key management infrastructure. They mentioned that the wireless channel between two nodes is a time-varying mapping. The mapping is also location specific, reciprocal and usually is called fading. By applying those principles, [5] was able to extract security keys. They measured the channel response and calculated the value of it. After that, they compared the calculated value with values that indicating channel statics to encode the value into either 1 or 0. Because the channel is time-varying thus the generated bit sequence would also vary according to time. Despite the fact that channel response is reciprocal the generated bit sequence might not be the exactly the same due to different kind of reasons, for example noise influence, calculation errors and so on. In order to extract the same part of the generated bits they used a way that we referred later in this paper to re-conciliate the bits. They were trying to find the central indices of bits that are 1’s or 0’s for certain length. Using the indices list they can come with key bits that are of high agreement rate.
[4]’s work examined the effectiveness of some previous works done by exploring the time domain with RSS measurement. They found that those solutions suffered some problems. Despite the fact that those solutions might generate relatively enough bits for security keys the entropy of the generated bits can be very low if there are not a lot of variations in the wireless channel under certain environments. An adversary is able to make the key generation to be predictable in static environments. A good thing about old solutions was that under dynamic environments the key bits can be generated fairly quickly with high entropy. In order to overcome the problems in static environments they modified the old solutions and came up with a solution that is able to adapt to different environments.

Those solutions focused more on the dynamic of channels over time domain and they suffered two problems: Low KGR and Mobility-dependent as [6] mentioned in his paper. Low KGR means that those solution will take a long period of time in order to generate long enough key bits. Mobility-dependent refers to the fact that if the receiver device stay static the performance of the solution will become poor in terms of secrecy.

2.2 Frequent domain exploration

In order to explore the dynamic over frequency domain some researchers applied the orthogonal frequency-division multiplexing (OFDM) model [6], [18], [19] to try to find new solutions for channel response measurements. [20][21] used phase change and [22] used multiple-input and multiple-output (MIMO) for exploiting channel dynamic over frequency-domain.

All those solutions except [6]’s suffers the impact of noise and might not have very ideal
results.

In [6]’s paper, by applying the principle of channel reciprocity combined with OFDM model he came up with a solution that explore the channel dynamic in both time and frequency domain, which significantly increased KGR. He used OFDM model to transmit a series of specially designed symbols repeatedly. By doing subtractions of the received symbols he was able to get rid of the influence of noise. His solution measured channel response at symbol level and had several advantages over traditional solutions. Traditional solutions would do the measurement at frame level and the channel may not vary very often. As a result, the measurements will not vary a lot and the generate keys might highly correlated with each other. In order to solve this problem, the two nodes must move to have temporal variant. Besides, due to the fact that old solutions will be affected by noise the measurements of channel response might not be very accurate and would impact KGR and BDR. Since [6]’s solution does the measurements at the level of symbol he was able to do subtractions to get rid of noise influence. Despite the fact that [6]’s solution can do a great job in generating security shared key, there is still room left for improving the KGR.
3 Methodology

3.1 A brief summary

Inspired by a previous work [6], we propose a solution that can achieve higher KGR, relatively low BDR and significantly high entropy. Traditional solutions for generating security keys was performed at the level of frames. In those cases, only one channel measurement can be taken from one frame. Those solutions actually waste channel resources that can be used for measurements because a frame contains many signal symbols that can be used for channel response. If we could obtain all of those signal symbols from the measurements, we would be able to increase our KGR to a significantly higher level.

3.2 Noise-free channel measurement

The principle of channel reciprocity tells us if transmitting the same frame Alice and Bob will be able to have the same channel measurements. In reality due to the influence of random noise that is caused by interference or thermal background it is not likely that Alice and Bob will have the same channel measurements. Noise will have significant influence over the channel measurements. In order to overcome this problem, I am using the same principle proposed by [6] to get noise-free channel measurements.

Channel response at time t is denoted as $h(t)$. Source symbol $s_t$ being transmitted with noise $n(t)$ will give us the received symbol as

$$r = h(t) \cdot s_t + n(t)$$

We can transmit source symbols with the pattern $(s, 0, \bar{s}, 0)$ repeatedly. Here we have $\bar{s}$ set
to be the conjugate symbol of \( s \) and 0 to be the silent symbol. The four received symbols corresponding to those four will be

\[
\begin{align*}
    r_0 &= h(\bullet) \cdot s + n(\bullet) \\
    r_1 &= h(\bullet) \cdot 0 + n(\bullet) \\
    r_2 &= h(\bullet) \cdot \bar{s} + n(\bullet) \\
    r_3 &= h(\bullet) \cdot 0 + n(\bullet)
\end{align*}
\]

The time needed to transmit 4 symbols is so small that we can assume within that time range of transmitting four symbols the noise will remain the same and the channel \( h(\bullet) \) will also not change. Based on that assumption we can get rid of the effect of noise by doing subtracting. We have

\[
\begin{align*}
    r_0 - r_1 &= h(\bullet) \cdot s \\
    r_2 - r_3 &= h(\bullet) \cdot \bar{s}
\end{align*}
\]

Base on the above two equations we can have the channel response denoted as

\[
h(t) = \frac{(r_0 - r_1)(r_2 - r_3)}{|s|^2}
\]

(2)

Since we know \( s \) for it is the symbol we are transmitting, therefore by measuring the received symbols’ amplitude we can have \( \frac{|r_0 - r_1||r_2 - r_3|}{|s|^2} \) representing the channel response. Every 4 consecutive symbols we can have one channel measurement. Since we are transmitting the pattern repeatedly and we will have a sequence like

\[
( s, 0, \bar{s}, 0, s, 0, \bar{s}, 0, s, 0, \bar{s}, 0, \ldots )
\]

Based on the same principles we mentioned above we can also have a valid channel response by calculating using symbols \( (0, \bar{s}, 0, s) \), we can also have a measurement done with \( (\bar{s}, 0, s, 0) \). In this case, a total of 12 symbols transmitted we can have 9 channel response measurements.

### 3.3 The ODFM inspired model

OFDM is wildly used in wireless communication nowadays. By using that model we are
able to exploit not only the channel dynamics over time domain but also over frequency domain. Since it is not possible for all of the subcarriers to be “0”s, inspired by OFDM piloting technique, [6] came up with a schema that can solve the problem. We present the schema below.

As Fig.2 shows we are transmitting \((s, 0, \bar{s}, 0)\) or \((0, s, 0, \bar{s})\) repeatedly over different subcarriers. This is a frame that contains \(N\) subcarriers and \(M\) symbols being transmitted. In reality, in order to get the maximum result, it is suggested to use \(M\) as big as possible in the standard.

Once the above frame is transmitted and received at the receiver, we denote the received symbol matrix as \(R = (r_0...r_{M-1})\)

\[
R = \begin{pmatrix}
    r_{0,0} & r_{0,1} & r_{0,2} & r_{0,3} & \ldots & r_{0,M-1} \\
    r_{1,0} & r_{1,1} & r_{1,2} & r_{1,3} & \ldots & r_{1,M-1} \\
    r_{2,0} & r_{2,1} & r_{2,2} & r_{2,3} & \ldots & r_{2,M-1} \\
    \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
    r_{N-1,0} & r_{N-1,1} & r_{N-1,2} & r_{N-1,3} & \ldots & r_{N-1,M-1}
\end{pmatrix}
\]

\[
= \begin{pmatrix}
    h_0 \cdot s + n_0 & h_0 \cdot 0 + n_0 & h_0 \cdot \bar{s} + n_0 & h_0 \cdot 0 + n_0 & \ldots \\
    h_1 \cdot 0 + n_1 & h_1 \cdot s + n_1 & h_1 \cdot 0 + n_1 & h_1 \cdot \bar{s} + n_1 & \ldots \\
    h_2 \cdot s + n_2 & h_0 \cdot 0 + n_2 & h_0 \cdot \bar{s} + n_2 & h_0 \cdot 0 + n_2 & \ldots \\
    h_3 \cdot 0 + n_3 & h_0 \cdot s + n_3 & h_0 \cdot 0 + n_3 & h_0 \cdot \bar{s} + n_3 & \ldots \\
    \vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}
\]

\(R\) is then partitioned into \(K=M-3\) sub-matrices as \((P_0, P_1...P_{K-1})\) with \(P_1 = (r_0, r_1, r_2, r_3)\) \(P_2 = (r_1, r_2, r_3, r_4)\) \(P_3 = (r_2, r_3, r_4, r_5)\)…\(P_K = (r_{k-4}, r_{k-3}, r_{k-2}, r_{k-1})\). We have assumed that the noise and channel
will stay still during 4 symbol times, and the measurements vector is then obtained as

\[
A_i = \begin{pmatrix}
|r_{0,i} - r_{0, i+1}| & |r_{0, i+2} - r_{0, i+3}| \\
|s|^2 \\
|r_{1,i} - r_{1, i+1}| & |r_{1, i+2} - r_{1, i+3}| \\
|s|^2 \\
|r_{2,i} - r_{2, i+1}| & |r_{2, i+2} - r_{2, i+3}| \\
|s|^2 \\
|r_{3,i} - r_{3, i+1}| & |r_{3, i+2} - r_{3, i+3}| \\
|s|^2 \\
|r_{4,i} - r_{4, i+1}| & |r_{4, i+2} - r_{4, i+3}| \\
|s|^2 \\
\vdots \\
|r_{N-1,i} - r_{N-1, i+1}| & |r_{N-1, i+2} - r_{N-1, i+3}| \\
|s|^2
\end{pmatrix}
\]

where 0≤i≤K-4

In this way, we can have channel measurements matrix denoted as \( A_{N \times K} = (A_0 \dots A_{K-1}) \) with \( N \times K \) elements extracted.

### 3.4 Key generation algorithms

Our method of generating keys is summarized below: first, Alice and Bob will transmit probing frames at each side using TDD. Once they individually have the frame received (the matrix A) the first thing to do will be Quantization to get a bit matrix. After that, each side will apply XOR operation for the bit matrix and they will have a queue containing the transcoded bits from the bit matrix as the output. Following the previous step they shall find the common parts between the queues generated at each side. Individually, they find consecutive 0s or 1s with length \( \lambda \) and keep the central index of that block in a list. Once the queue is gone through, at Alice’s side we will have a list \( L_A \) and \( L_B \) at Bob’s side. The list is then transmitted to each other to calculate the common part. At both sides, they shall have \( L_{AB} = L_A \cap L_B \). Using this indices list with the bit queue they generated separately at each side, they are able to generate the secret key independently.

Following is the details about each step:
3.4.1 Step1: Quantization

| **Input:** channel measurement matrix $A$ of size $N \times K$ |
| **Output:** A bit matrix $A$ of size $N \times K$ |
| **Procedure:** |
| 1. Calculate the mean of each column, denote the value as $\text{avg}_k$ for $k$-th column. |
| 2. For each element in a column compare its value with $\text{avg}_k$. If that value is greater than $\text{avg}_k$ denote element at that index as 1 else as 0. |
| 3. Repeat for every column. |

With the above quantization algorithm, for an input measurement matrix we compare each element’s value with the average value of the column it belongs. We are comparing over different subcarriers. By comparing the values we can encode each numeric value into a binary value. We will have a bit matrix as the output.

3.4.2 Step2: XOR operation

| **Input:** A bit matrix $A$ in step 1 of size $N \times K$ |
| **Output:** A bit queue with $(N-1) \times K$ elements, a XORed bit matrix |
| **Procedure:** |
| 1. For all elements in a column do exclusive-or repeatedly and consecutively. |
| 2. Repeat for every column to get a XORed bit matrix. |
| 3. Put each column into a queue according to the column order to get a bit queue. |

The above algorithm we take the bit matrix in step1 as input. For each column, if we denote the elements as $c_0$, $c_1$, $c_2$, $c_3$...$c_{n-2}$, and with the XOR operation we will have $c_0 = (c_0 \oplus c_1)$, $c_1 = (c_1 \oplus c_2)$...$c_{n-3} = (c_{n-3} \oplus c_{n-2})$. For a total of $N$ inputs we will have $N-1$ outputs. We repeat the operation for all the other columns and will have a bit matrix of $(N-1) \times K$ elements as output. The reason for doing the XOR operation is that we can transform ...0101… sequence into 1s
and sequence like …000… or …111… into sequence of 0s. We want consecutive 0s or 1s in order to generate security key, which we will mention soon, and this operation will let us make full use of the bits matrix generated in step 1. We also put the XORed values into a queue according to the column order. We will have a queue containing the content of the bit matrix. Later, we will use the indices of elements in the queue to generate the security keys.

### 3.4.3 Step 3: Bit Reconciliation

<table>
<thead>
<tr>
<th>Input: XORed bit matrix in step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: a list of indices</td>
</tr>
<tr>
<td>Note: Each bit in the bit matrix is denoted as $e_{n,k}$</td>
</tr>
</tbody>
</table>

**Procedure:**

1. For each column find consecutive bits of length $\lambda$.
2. Starting element is denoted as $e_{s,k}$ and end element is denoted as $e_{e,k}$, thus central index is denoted as $\left[\frac{s+e+2k(N-1)}{2}\right]$. Put it into a list.
3. Repeat for all columns.

We use the XORed bit matrix as the input and each element in that matrix is denoted as $e_{n,k}$, where $k$ is the $k$-th column and $n$ is the $n$-th element in that column. For each column, we try to find a sequence of $\lambda$ 0s or 1s. The starting element of that sequence is $e_{s,k}$ and the ending element is $e_{e,k}$. The central element of that sequence is then denoted as $\left[\frac{s+e+2k(N-1)}{2}\right]$ as this to be the same index of that in the bit queue we generated in Step 2. If we have four consecutive elements to be the same and we have $\lambda$ to be 3 we shall have 2 central indices form that sequence. By doing this operation separately at each side, Alice and Bob will have a list of indices, namely $L_A$ for Alice and $L_B$ for Bob.
3.4.4 Step4: Key Generation

<table>
<thead>
<tr>
<th>Input: Lists of indices from step3 and the bit queue from step2.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: security key</td>
</tr>
<tr>
<td>Procedure:</td>
</tr>
<tr>
<td>1. Both side send the list of indices to each.</td>
</tr>
<tr>
<td>2. Each side do intersection for the received indices list and the list itself generated.</td>
</tr>
<tr>
<td>3. Use the intersection indices list with bit queue mapping to get security key.</td>
</tr>
</tbody>
</table>

Once Alice and Bob sides all have their list of central indices ready, they will send the list to each other. Alice and Bob will have two lists of central indices and they will do intersection operation individually. After the operation, they will have the same list of central indices as $L = L_A \cap L_B$. By using $L$ they can independently generate security key with the bit queues form step2: just map the indices in the list with the indices of elements in the queues and they will have the bits for security keys. The only information Eve can get during the whole process is the list of indices that was transmitted to each other. Due to the fact that Eve does not have the bit queue, the list of indices will just be some useless numbers for him.

3.5 A case study of $\lambda$ choice

In the previous work done by [6], he has proved that when $\lambda = 3$ we can have the optimal result. Since we are doing the testing over Matlab we want make sure that condition still holds.

We did a sample run with SNR set to be 15 and nose difference to be 5%, with other conditions set to be fixed and we had the result as the table below.
One thing to notice from the table is that we did not include $\lambda=1$ because when $\lambda=1$ we will have very high BDR and as a result the generated keys cannot be used for encryption. As we can see from the table, with the increase of $\lambda$ the overall entropy of the generated bits does not change significantly and they all kept a significantly high value. Different values of $\lambda$ will affect bit disagree rate and key length significantly. When $\lambda=5$ or 6 we had bit disagreement to be 0 which is very good. But this will lead to the problem that the number of generated bits is quite small and it hurts the KGR. This happened because with larger $\lambda$ we need more consecutive 0’s or 1’s and that situation might not happen very frequently in reality. Since long sequence of 0’s or 1’s cannot be generated very frequently it is not very likely to have them mismatched. For that reason, the bit disagreement rate will be very low or even 0.

For $\lambda=2, 3, 4$ our solution tends to have a pretty good amount of generated key bits. Comparing $\lambda=3, 4$ we had bit disagreement reduced by a difference of 0.0121 but the bits generated was reduced by 180.2 bits. Overall speaking, set $\lambda$ to be 3 will be a better choice than 4. For $\lambda=2, 3$ we can see that even the bits generated when $\lambda=2$ is 1.38 times more than that of 3, the bit disagreement rate is about 3.26 times higher. Based on the results above, we would like to choose $\lambda=3$ for our test setting, which is the same number [6] used in his experiment.

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Entropy</th>
<th>Bit Disagree Rate</th>
<th>Key Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.9984</td>
<td>0.1114</td>
<td>730.4000</td>
</tr>
<tr>
<td>3</td>
<td>0.9949</td>
<td>0.0261</td>
<td>306.2000</td>
</tr>
<tr>
<td>4</td>
<td>0.9883</td>
<td>0.0140</td>
<td>126.0000</td>
</tr>
<tr>
<td>5</td>
<td>0.9512</td>
<td>0.0000</td>
<td>58.8000</td>
</tr>
<tr>
<td>6</td>
<td>0.9277</td>
<td>0.0000</td>
<td>23.7000</td>
</tr>
</tbody>
</table>

**Figure 3: A case study of $\lambda$.**
4 Performance evaluation

In order to see how effective our solution is we tested it by using Matlab under three
different noise models. The symbol s we used was $1+j$ for the simplicity of calculation. The
value of $\lambda$ we chose was 3 because it can optimize the result in terms of key generation.

4.1 Set up

We used Matlab to transmit frames between Alice and Bob. The frame has 200 subcarriers
and the number of symbols per carrier is 12. We used 16QAM and a total of 1200 bytes of data
per frame was transmitted. We added noise to the frame to simulate the transmission in a real
channel environment. In order to mimic reality but also keep the channel reciprocity property
we designed the noise in a way that it can be changed with a certain ratio of difference. The
difference was also a criteria that we examined. In that case, one noise added to the frame
“transmitted” to Alice will be partially different from the noise added to the frame “transmitted”
to Bob. They will see different transmitted frames but without a very big difference. This
practice can keep the property of channel reciprocity and, due to the fact that the ratio of
difference is configurable, we can test our solution under some extreme conditions such as a
huge difference of the noise, which may sometimes happen in reality.

Because our solution is based mostly on [6]’s work we would like to compare our result
with his solution to see how effective our solution is. The 3 different noise models we used
were: White noise model, Rayleigh noise model and Ricean noise model. The 3 criteria that
we used to compare the results were: entropy Rate, Bit Disagreement Rate and Key Number
per 100Bits. Entropy rate refers to how random the generated keys are and with a higher
entropy rate the generated key is less likely to be hacked by using statistic methods. Bit Disagreement Rate also plays an important role in measuring the quality of the generated keys. A high BDR value will make the generated keys impossible to use. Since our solution was measuring the channel response at the level of symbols, how many bits we transferred pre frame will also have an effect on the result; thus, we have decided not to measure how many key bits a frame could generate but how many key bits can be generated for a total of 100 bits transmitted. In reality, different people may have different frame sizes and this measurement can be a fairer way to examine the effectiveness of key generation.

4.2 White noise model

4.2.1 Noise difference 5%, SNR vary

Our first experiment was done under the white noise model, which simulates the random noise channel. First, we set the noise difference to be 5% and observe what the results could be for different SNR values. The following tables show the result.
From the tables above we can see that with different SNR values our solution will have almost the same entropy as the old solution. In fact, for some of the SNR values the entropy value generated by our solution is slightly higher than those of the old solution. For BDR, we can see that our solution is almost performing the same as the old solution. When noise difference is set to be 5% our solution and the old solution will have a BDR between 1% and 3%. We can see that BDR is centered on 2% for both solutions. In terms of Key bits generated per 100 bits, our solution is always higher than the old solution. For the simplicity of comparison we used the ratio between our solution and the old solution to demonstrate how effective our solution could be. As the last table shows in Fig.4 the ratio is always about 3 which matches the analysis we mentioned before: for 12 symbols transmitted our solution can have 9 channel responses and the old one can only have 3. One thing we found interesting was that different SNR values do not play a very significant role in the effectiveness of our solution under the white noise model. As we can see from the tables above: different SNR values just slightly change the 3 criteria we were focusing on. This makes sense because we assumed that noise will not change during 4 symbol times and we can do subtractions to get rid of the noise.
influence. This result proved that our assumption of invariant noise fits in the white noise model.

4.2.2 Noise Difference 10%, SNR vary

The second test we did under the white noise model was to set the noise difference to be 10% and vary SNR values to observe the result. The result is shown with the figure.

![Figure 5: Performance under White Noise Model with 10% noise difference](image)

From the entropy chart we can see that the generated bits’ entropy is still very high for both solutions just like the previous one. The entropy for the old solution seems to grow with the increase of SNR. Since the value of entropy for the old solution is already very high the increment might just be a coincidence. Comparing the entropies for both solutions, we can see
that our solution will have a higher value. The value for both solutions is already very high so we can just say that both solutions did well in terms of entropy rate. When it comes to the BDR we can easily observe that our solution will have a slightly higher value for some SNR values and a slightly lower values for the others. The values for both solutions centered on 5%. In general, our solution had as good a performance as the old solution. As part3 of Fig.5 shows, the number of key bits generated per 100bits of our solution will always have a higher value across different SNR values. The last part of the figure shows that the ratio of the key bits number generated by two solutions always floated around 3, which is the same as what we had expected. One important thing to notice is that even when the SNR value changed, the results kept stable. In other words, SNR does not play a very important role in this noise model, which is the same as we have observed.

4.2.3 A brief comparison of the two experiments

One interesting thing we observed by comparing the entropy values of both test sets is that the entropy for the old solution seems to fluctuate more than our solution. The entropy of our solution tends to be a more flat line while the old solution has more waves. With the increase of noise difference from 5% to 10% we can see that the BDR also increases. As with 5% noise difference we have the BDR centered around 2% but for 10% the difference grew to 5%. A similar pattern can be observed with the key bits number generated per 100 bits: the number was 3 for our solution but dropped to about 2.4 with the increase of the noise difference.

4.2.4 SNR 25, noise difference vary between 5% -35%

As we have seen before, different SNR values will not significantly affect our result but
the noise difference will. We then fixed the SNR value and tested how effective our solution was in terms of various noise differences. For this test, we set the SNR value to be 25 and noise difference varied form 5% to 35%.

The results are shown in the following figure:

**Figure 6: Performance under White Noise Model with SNR 25dB**

We can see that for a fixed SNR value (25) with the increase of noise difference the entropy, BDR and key bits number per 100 bits all changed accordingly. As the difference increases, the entropy for the old solution tends to go down. Fig. 6 tells us that the entropy of the old solution tends to drop faster at a noise difference around 20%. On the contrary, the entropy of
our solution seems to not change very much. Even at the point when the old solution is dropping faster our solution can still keep a high entropy value. As we have expected, the BDR increases and the key bits number per 100 bits drops with the increment of noise difference. This happened because as with the growth of the noise difference between Alice and Bob, the difference of the received frames also grew. As a result, it will cause more bits of the generated key to be different from each other causing the BDR to grow.

Despite the fact that both solutions tend to have a worse performance over a high noise difference our solution can still do a good job in terms of generating security key bits. Part 4 of Fig.6 shows the ratio of key bits generated by our solution is still about 3 times the old solution. At the same time, our solution keeps the entropy no lower than the old solution and BDR no higher than the old one.

4.3 Rayleigh noise model

The Rayleigh noise model simulates the condition in which wireless signals propagate among high buildings. There is no Line of Sight (LOS) between the sender and the receiver. In other words, the signal cannot directly propagate from sender to receiver. Instead, the receiver gets the signals only from the reflections of the signals over different objects.

4.3.1 Noise difference 5%, SNR vary
As we can see from Fig. 7, the entropy value of our solution centers on 0.99 and the old one is 0.98. Different from the white noise model, our solution’s entropy is stable but the old solution’s fluctuates, the entropy rate for both solutions under the Rayleigh noise model seems to change synchronously. As with the increase of the entropy of the old solution our solution’s entropy also increases and they seems to have a similar tendency of change over different SNR values. The BDR of the two solutions all centered on 4%, which is two times the amount of that under the white noise model. Comparing the results of both solutions we can come to the same conclusion: both solutions did as well as each other. The number of bits generated per 100bits is about 2.5 in our solution which is about three times the old solution. Under the white noise model, with 5% noise difference, we had that value to be 3. We can say that both solutions work better under the white noise model. The SNR value also does not have a big impact on the results of both solutions which is the same as the previous noise model.
4.3.2 Noise difference 10%, SNR vary

For this test, we set our noise difference to be 10% and compared the results over different SNR values. We want to see what the results will be under this test setting.

![Graph](image)

**Figure 8** Performance under Rayleigh Noise Model with 10% noise difference

Fig.8 part1 shows the entropy value comparison between our solution and the old solution. As we can see from the figure, the entropy value of our solution is always bigger than the previous one despite the fact that both solutions already had significantly high values. The tendency of entropy change over different SNR values is similar for both solutions, which is the same as the 5% noise difference setting. The entropy of our solution was around 0.99 but the old solution’s dropped a little bit under the 10% noise difference, which might be a
coincidence. We can see that our solution does not have an advantage over BDR. By comparing the values we found that our solution has a relatively higher BDR rate than the old solution under the Rayleigh noise model. For example, the BDR value of our solution is about 2 times the value of the old solution when SNR equals 30. But this seems to be just a coincidence because for all other SNR values our BDR is just slightly higher in terms of the percentage value. Despite the fact that our solution does not have an advantage over BDR value, we did a good job in generating key bits. As SNR grew from 19 to 30 our solution had a value around 2.5 but the old solution only had a value of about 0.8. As the last table clearly shows, the ratio is about 3. This means our solution can generate about 2 times more bits than the old solution under the Rayleigh noise model. Like the previous noise model, the SNR value also is not a major factor under this noise model.

4.3.3 A brief comparison between tow noise differences

With the change of noise difference from 5% to 10% we didn’t see a big difference in terms of entropy rate. Both solutions did pretty good job under the two noise differences settings. The BDR didn’t change a lot just with slight differences. We can see that under 5% noise difference setting we would have less values whose DBR value is over 5% and more under 10% noise difference. The number of bits generated seemed to have the same value for both noise difference. If we fix the SNR value and change noise difference, the difference will become apparent.

4.3.4 SNR 25, noise difference vary between 5% -35%

Another test we did for Rayleigh noise model was that we set the SNR value to be 25 and
observed the results under various noise differences.

Under Rayleigh noise model, when we set the noise difference to be variant we can observe similar results under white noise model. With the increase of noise difference, the entropy of key bits generated by both solutions all kept high values. One thing can be observed from the above table is that our solution seems to be more stable in keeping high entropy of the generated key bits. Larger noise difference will make BDR to increase and key number per 100 bits to decrease which is that same as the previous noise model. Despite all those changes, our solution was still able to generate 2 times more key bits as the ratio chat shows. Overall speaking, our solution under Rayleigh noise model performed as well as that under
white noise model.

### 4.4 Ricean noise model

The Ricean noise model is similar to the Rayleigh noise model except a small difference. Compared with the Rayleigh noise model where there is no LOS between the sender and the receiver, Ricean noise model includes LOS. In other words, the signal can propagate directly from send to receiver and the receiver also receives signals from other paths through reflections at the same time.

#### 4.4.1 SNR 25, noise vary between 5% -35%

We will start with SNR value set to be 25 and vary the noise difference from 5% to 35% as we did before.
Noise difference (%)

Figure 10: Performance under Ricean Noise Model with SNR 25dB

As Fig.10 shows that, the overall result is the same as that under previous two noise models. One thing we want to mention is that under this noise model the entropy of key bits generated by both solution is lower than the previous two noise models. Our solution and the old solution had entropy values around 0.9. It seems that SNR value might have some influence under this noise model. We will show in the following test sets that SNR value do play an important role under this noise model. Under this setting, with the increase of noise difference BDR goes up and key number per 100 bits decreases accordingly. Our solution was still able to keep a good ratio of key bits generated with the old solution.

4.4.2 Noise difference 5%, SNR vary

As we have mentioned before, SNR value might has impact on the results of both solutions. The results are summarized with the following figure.
As we can see from the figure that with the increment of SNR value the entropy for both solutions drop. For SNR value between 19 and 25, the entropy value dropped 0.1 and 0.2 between 25 and 30. We can see that when SNR value equals 25 the entropy value started to drop faster for both solutions. Comparing the results of the two solutions in terms of entropy, we can come to the conclusion that they are almost identical to each other. When it comes to BDR we can observe that with the increment of SNR we will see a drop in BDR. Our solution still has the same performance as the old solution for the BDR criteria. A possible explanation for the DBR decrease with SNR increase could be that noise might not be stable during 4 symbol times. As a result, the subtractions might not get rid of the noise influence perfectly. The results were still polluted by the noise somehow. When we increase the SNR, we will have a stronger signal strength and the noise influence will be relatively smaller. Thus, we can observe the tendency of BDR decrease. As with the increment of SNR, the key bits generation rate also increases. The value was boosted form 3 to about 6 for our solution and 0.9 to 2.0 for the old one. Our solution still kept the key bit ratio at 3 for this test setting.
4.4.3 Noise difference 10%, SNR vary

In order to investigate how different SNR values will affect the result under this noise model we set noise difference to be 10% and change SNR value from 19 to 30 to see the different results.

![Graph showing entropy change with SNR](image1)

Fig.12: Performance under Ricean Noise Model with 10% noise difference

Fig.12 shows the entropy, BDR, key number per 100 bits with the change of SNR value. As the SNR value goes up, the entropy of both solutions go down. For SNR value changed from 19 to 30 entropy went down from 0.95 to 0.63 and this tendency was observed in both solutions. By comparing the values in details we can see that our solution had entropy slightly higher than
the old one. BDR goes down with SNR goes up which is shown in Fig. 12. We have BDR as high as 0.1243 at SNR equals to 19 and the value is 0.0324 when SNR is 30. The BDR is roughly the same for both solutions. For key bits number generated by both solutions we are able to see the same result: our solution had about 2 times more bits generated. Overall, both solution did just fine under the Ricean noise model because entropy will drop with SNR increment.

4.4.4 A brief comparison of two noise difference

As we can see, by comparing Fig.11 and Fig.12, the entropy change tendency is very similar to each other for noise difference of 5% and 10% under the Ricean noise model. They all dropped and dropped faster at SNR around 25. The biggest difference took place at the BDR value. For 5% noise difference we have the value to be about 5% but increased to 11% for 10% noise difference. For other SNR values, BDR difference between two tests also showed big differences. We can say that under Ricean noise model our solution is very sensitive to the variation of SNR value as well as noise difference value.
5 Conclusion

This paper came up with a better solution for generating security keys between two wireless nodes by applying the principle of channel reciprocity. The solution was able to improve the KGR significantly compared with [6]’s solution. In our tests, we were able to increase our KGR by 2 times more than [6]’s solution without at the cost of having lower entropy rate. In fact, the entropy generated by our solution is slightly higher than [6]’s solution. The fact is that as both solutions already had high enough entropy it is not necessary to compare which one does better over entropy values. For KGR, if more symbols could be transmitted under our solution we would have a higher ratio of more bits generated. The ratio could be calculated as \((x-3)*4/x -1\) where \(x\) is the number of symbols that greater or equal 4. When \(x\) is 12 we will have \((12-3)*4/12\) which equals to 2. The ratio limit will be 3 when \(x\) is large enough. There was a little problem with our solution: if we had more bits generated we would also have higher BDR sometimes. Further work is needed to bring down BDR more effectively.
Reference


