Reconfigurable Convolutional Codec Architecture and its Security Application

Liang Tang[1] Jude Angelo Ambrose[1] Sri Parameswaran[1]

[1] University of New South Wales, Australia
{liangt, ajangelo, sridevan}@cse.unsw.edu.au

Technical Report
UNSW-CSE-TR-201403
February 2014

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

Wireless communication is an indispensable tool in our daily life. Due to the open nature of wireless channels, wireless communication is more vulnerable to attacks than wired communication. Security is paramount in wireless communication to overcome these attacks. A reconfigurable convolutional encoder/decoder based physical layer security mechanism, named *ReConv*, is proposed in this paper. *ReConv* provides an extra level of security at the physical layer by dynamically updating baseband convolution parameters for secure packets. *ReConv* is expected to interleave normal packets along with secure packets. An eavesdropper will see the packets with changed convolution parameters as packets containing errors and will drop them. However, the rightful receiver will be able to decode the secure packets without error. Since the secure packets are dropped at the eavesdropper's physical layer, there will be no further processing of the secure packets by the eavesdropper. *ReConv* allows the use of 63.5 billions differing convolution parameter combinations; the attacking complexity would be increased exponentially with the extended byte-level *ReConv*. Meanwhile, the low hardware overhead of *ReConv* makes it a low cost solution to enable greater security in existing wireless systems. Furthermore, the orthogonal relationship between *ReConv* and existing security algorithms such as AES, makes it easy to integrate *ReConv* to existing wireless systems.

1 Introduction

The revolution in wireless technology has created a plethora of applications, ranging from consumer electronics to highly sensitive military applications. Such applications often need to perform secure transactions over a public medium. It has become common practice to use cryptographic protocols, such as AES, ECC, etc. at the MAC layer or at the application layer, to encrypt the data before transmission. Even though these encryption techniques protect information, the actual data transmitted through the wireless medium (i.e., physical message stream, referred to as *physical message* from this point onwards) can be easily captured by an attacker, since the wireless medium is unprotected. Successful capture of the physical message will allow attackers to leisurely explore security vulnerabilities in the encryption, and use these vulnerabilities to extract the secure information. For example, eavesdropping on Bluetooth links [1, 2], and side channel attacks [3, 4]have been quite successful in recent years.

The physical layer security technologies protect the physical messages from adversaries by reducing the chances of packet capture. Thus, any possibilities of offline attacks would be completely eliminated. One of the methods used is frequency hopping. For example, in Bluetooth Adaptive Frequency Hopping (AFH) mode, there are 79 channels one can use to send a packet [5]. By hopping channels according to a pattern known to only the transmitter and the legitimate receiver, an eavesdropping adversary can be confounded.

Frequency hopping comes under the broader umbrella of spreading. Spreading [6] in wireless protocols has an inherent ability to hide the physical message, however, it is vulnerable to cross correlation analysis [7]. Spreading at most has a few thousand possible ways in which to send a message, depending on the size of the spreading code [8].

Custom secret modulation mapping schemes were attempted in [9] as another physical layer security mechanism. Even though this would confuse the adversary, an exhaustive attack on the physical message to find the correct modulation mapping scheme is not difficult, since there are very few modulation mapping schemes. The number of modulation schemes used to confuse an adversary in [9] is $4!$ in the QPSK modulation scenario.

A function block diagram illustrating the Wi-Fi physical layer is given in Figure 1.1. In current literature, there exist proposals to change the parameters associated with the spreading ([10]), modulation mapping ([9]), and puncture ([11]) signal processing function blocks for security applications. For the first time, in this paper we propose a method to dynamically change the parameters of the convolutional encoder and the corresponding receiving component, Viterbi decoder, in order to make the exhaustive analysis harder. The convolutional encoder and Viterbi decoder pair are primarily used in wireless communication for error correction. Viterbi decoder is a complicated component in wireless protocol signal processing systems. Usually the convolutional encoder unit and the corresponding Viterbi decoder have fixed parameters. In our system, these units are flexible, and their parameters can be changed rapidly. These changes, allow approximately 63.5 billion possible parameter combinations. Creating a complicated run-time reconfigurable convolutional encoder and decoder pair enforces a much more complicated decoding strategy, making an exhaustive attack on the wireless channel very time consuming. Such a finer switch within a protocol will make the prediction of the parameters harder by many folds.

We are proposing a security engine, *ReConv*, which will utilize a fast switching reconfigurable convolutional encoder and Viterbi decoder to change convolution pa-

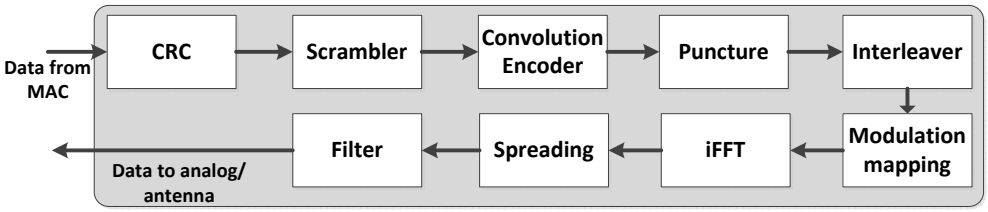

Figure 1.1: PHY layer signal processing function blocks

rameters at runtime. Our *ReConv* engine is tested on Wi-Fi (for proof of concept) and can be applied to any wireless protocol which utilizes convolutional encoding as an error correction mechanism.

Outline

The rest of this paper is organized as follows. In section 2, the current literature on physical layer security is reviewed. Section 3 provides the methodology of *ReConv* based secure communication. Section 4 and section 5 show the experimental setup and results. The conclusion is given in section 6.

2 RELATED WORK

Currently, there exist techniques to provide security at the physical layer of wireless communication, such as spread spectrum technologies [6, 12, 13, 14, 10], modulation mapping constellation technologies [9], and eavesdropper channel degrading techniques[15, 16, 17].

Spread spectrum techniques are commonly utilized in the physical layer of a wireless communication system to disperse the information over a range of frequencies, thereby increasing the transmission bandwidth while reducing the interference from the other signals [18]. Realising the actual information bit using a secret spreading pattern is an effective way to hide the information over wireless channels. Direct sequence spread spectrum (DSSS) approach [10] spreads a low bandwidth signal over a broad frequency range. Hence the low power signal is obfuscated by the noise in the medium. Frequency hopping spread spectrum (FHSS) [6, 19] is an improvement for spread spectrum to address security, where the wireless signal is transmitted by hopping to different frequencies. Both the transmitter and receiver switch frequencies, either, when an attack is detected, or continuously to confuse eavesdroppers. The hopping sequence is shared between the transmitter and the intended receiver before the transmission as a secret key. Bluetooth uses a pseudo-random frequency hopping (PFH) [12] to further obfuscate the dependency between communication and secret key establishment. PFH allows the transmitter and receiver to randomly hop to different frequencies and exchange the key when they happen to operate at the same frequency, which will be at a random unpredictable instance [20]. Ling and Li [13] modelled the spread spectrum to handle jamming attacks, where an adversary can obstruct the spectrum signals. Martin et al. [14] proposed a Hybrid Spread Spectrum (HSS), combining FHSS and DSSS. HSS increases the complexity of decoding/despreading for the eavesdropper, by changing the length of the data through the DSSS, while hopping frequencies through the FHSS.

A modulation mapping constellation approach is presented in [9], where the transmitter and receiver use a secret custom constellation mapping. Modulation involves converting the bit sequence real and imaginary components using a constellation diagram. To decode the message correctly, the receiver needs to know the type of modulation used as well as the mapping structure of symbol to bit sequence (i.e., constellation diagram). Hence, the authors in [9] created a custom secret constellation mapping scheme by changing the relationship between constellation points (i.e., real and imaginary) and bits. For example, 16QAM can use circular constellation or rectangular constellation. The mapping from bit sequences to constellation points will have $M!$ variations for a M-symbol modulation constellation. The value of M can be a small number in certain modulation scheme. For example, M is only four in the QPSK modulation. Thus, the modulation mapping constellation approach can be easily broken.

A reconfigurable puncturing pattern is proposed to encrypt the communication data in [11]. Puncture is a signal processing block after the convolutional encoder or the Turbo encoder to intentionally remove certain encoded bits, in order to improve the data rate. The receiver appends dummy bits according to the puncturing pattern so that the convolutional decoder or Turbo decoder circuits can be identical for both punctured and non-punctured data. All appended dummy bits can be classified as error bits, however, due to the error correction ability of the convolutional decoder and the Turbo decoder, these errors can be recovered. A normal random sequence is generated to control the puncturing pattern in [11]. When the eavesdropper doesn't know the puncturing pattern, the received packet cannot be successfully decoded.

Phased array [21] enables secure wireless transmissions by enhancing signal transmission in the direction of the desired receiver, while degrading the signal in most of the other directions. Such directional transmission requires an adversary to use sophisticated sensitive receivers to recover the suppressed signals. To address such a vulnerability, Babakhani et al. [22] proposed a near-field modulation technique with a reflector switching antenna, which would generate the correct signal constellation only in the desired direction. The signals for other directions were scrambled using a parasitic array based transmitter. Daly and Bernhard [17] proposed a similar technique using phased arrays. Chorti et al. [15] proposed to degrade the eigenvalues of the eavesdropper's channel, compared to degrading the eavesdropper's SNR, by injecting noise into the channel. Chorti [16] further proposed to inject jamming signal to degrade the eavesdropper's channel, than using plain noise to improve the security.

Similar to the approaches introduced in this section, we propose to share a secret key to change the physical layer signal processing, but with more parameter combinations to increase attacking complexity. The modulation mapping constellation approach [9] always use a fixed (but secret and custom) constellation diagram. Hence exhaustive search is feasible by the adversary, since [9] only has $8!$ variations. Spreading techniques can hide the information in a pseudo-random spreading code, but intensive correlation analysis can find the spreading code. For example, only 2^{11} correlation calculations are needed to estimate the spreading code used in the 802.11b Wi-Fi standard. The channel degradation techniques [15, 16, 17] can only reduce the possibility of an attacker capturing the signal, however, sophisticated equipment can be used to get around channel degradation. The number of parameter combinations possible in our proposed security mechanism is tremendously larger compared to existing physical layer security approaches. Furthermore, the time consuming Viterbi decoding algorithm must be executed to try each parameter combination; this feature improves the attacking complexity in another order. We also propose to randomly change the parameters during transmission and to interleave the secure packets and the normal packets

together, hence, making the exhaustive approach almost impossible. We are assuming the existence of a safer secret key exchange mechanism as proposed in [12].

3 The methodology of ReConv Based Secure Communication

The methodology of secure wireless communication based on the dynamic reconfigurable convolutional codec architecture, *ReConv*, including the *ReConv* encoder and the *ReConv* decoder, is presented in this section. *ReConv* encoder is an iterative parallel LFSR based reconfigurable convolutional encoder, and *ReConv* decoder is a reconfigurable parallel Viterbi decoder.

Convolutional encoder and its corresponding decoding function block, Viterbi decoder, are generally utilised in wireless communication systems to correct the communication error with the redundant coded bits. Convolutional encoder converts each 1-bit information symbol into an n-bit coded symbol, where $1/n$ is the coding rate R. The transformation is a function of the last K information symbols, where K is the constraint length. Convolutional encoder is designed using a Linear Feedback Shift Register (LFSR) structure, with additional XOR operators. The number of output branches equals to n, and the inputs of XOR operators are defined by octal polynomials g_i for the i^{th} branch. The Wi-Fi convolutional encoder is depicted in Figure 3.1 as an example. Please note that the relationship between the g_i and the inputs of XOR is revealed by the dotted lines.

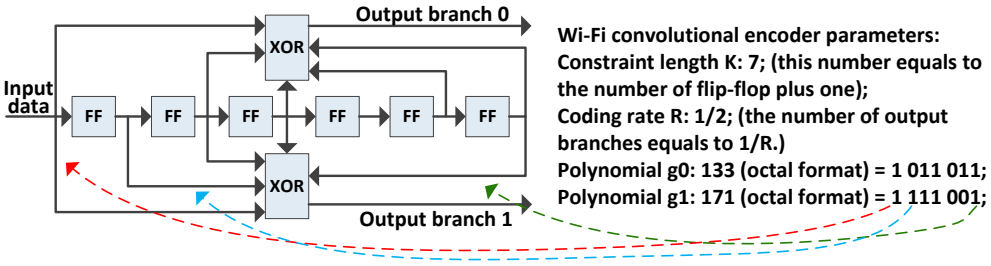

Figure 3.1: Wi-Fi convolutional encoder

3.1 ReConv Secure Communication

The secure reconfigurable convolutional codec architecture consists of a circuit based dynamic reconfigurable convolutional encoder at transmitter side, and a reconfigurable Viterbi decoder at receiver side. When the secure information needs to be sent, both the convolutional encoder at the transmitter and the Viterbi decoder at the receiver are set to a new set of predefined parameters. We assume a proper mechanism in place to have this mutual switch using upper layers, for example, session keys are exchanged between the transmitter and receiver offline. Since our circuit based reconfigurable architecture is able to achieve ultra-fast switching, the normal communication and the secure communication can be interleaved. Such interleaving hides the secure communication within the normal communication.

Figure 3.2 shows a secure communication scenario interleaving with normal communication, which is video transmission. Transmitter sends the video payload packets

to receiver using the Wi-Fi protocol, setting parameter q for the convolution codec. These video packets are shown as single (blue color) vertical arrows in Figure 3.2. Both the receiver and eavesdropper can successfully decode these packets at the physical layer, checked by the CRC (marked as a "tick" in Figure 3.2). Transmitter changes the convolution parameters to $q\prime$ when the secure information is to be sent. Receiver knows the timing to change the convolution parameter, while eavesdropper does not (note that there is an exchange mechanism in place from upper layer as described below). Thus, receiver can decode the secure packet correctly. Eavesdropper's receiving circuit still decodes the received signal using parameter q, and a CRC error will be observed by the eavesdropper to indicate a receiving error. Since the eavesdropper would still assume that the protocol is Wi-Fi, the eavesdropper would classify this error as occurred due to the wireless channel noise or interference. The upper layer protocols at the receiver will despatch all video packets to video App, while all secure information packets are despatched to a secure text App.

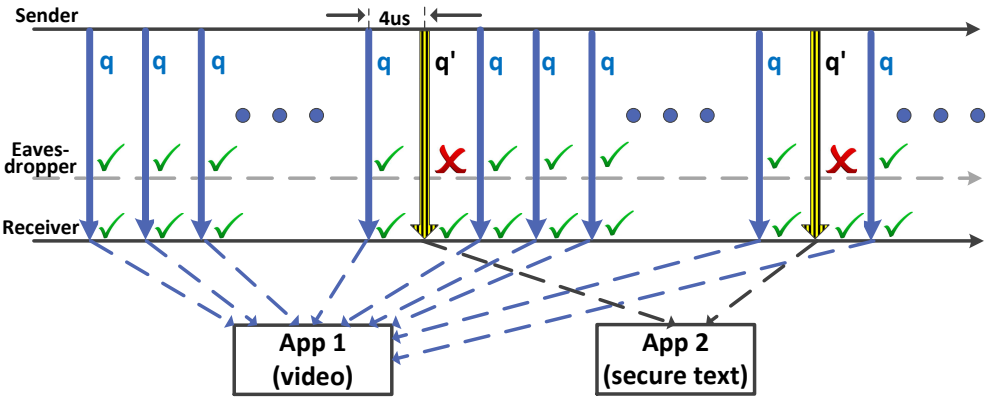

Figure 3.2: Reconfig convolutional codec secure scenario

The transmitter and receiver exchange a secret key offline (a common standard practice in wireless applications [5]). We propose the secret key (S), which is a set of parameter combinations, for our solution as shown in Equation 3.1. P is the time defining when the next secure mode will be activated. We suppose that the sender and receiver sharing a common clock, such as the GPS atomic clock. The $g0$, $g1$, $g2$ and $g3$ are polynomials for branches 0, 1, 2 and 3 represented by octal numbers. The constraint length, K, can be derived by the highest valid bit position of polynomials. For example, the value of K in Wi-Fi can be concluded as seven through its polynomials defined in Figure 3.1, because the highest valid bit position of both g0 and g1 is six. The coding rate, or the number of output branches, is not included in the secret key (S), since it is directly decided by the polynomial. The g_i can be set to zero to disable the corresponding output branch. For example, both g2 and g3 are zero in Wi-Fi, since the Wi-Fi coding rate is 1/2.

$$\begin{aligned} S &= \{P, G\} \\ G &= \{g0, g1, g2, g3\} \end{aligned} \tag{3.1}$$

Algorithm 1 depicts the flow of the transmitter and the receiver during secure and normal communication. Aligning with the standard practice in wireless communica-

tion, we allow the transmitter and the receiver to use the same pseudo random number generator (*PRNG*) which will generate the same random number from a given seed (*SD*). The *SD* is exchanged offline with the S, as well as the *PRNG* between the transmitter and receiver. Both transmitter and receiver will choose the initial set of parameters C_i from S. The seed *SD* will be utilised for the first time the *PRNG* creates a pseudo random number. Iterations after that will align at both ends to create the same pseudo random numbers at every iteration. This will enable choosing the exact same parameters at both ends at a particular switching instance. The convolution parameter for the current secure packet is selected by the *SEL* function and configured by the *SET* function. The procedures for convolution parameter setting are the same at both the sender and the receiver. The receiver waits for the secure packet with the alternated parameter for a predefined waiting time window. The receiver will exit to normal receiving, after either receiving a valid secure packet, or timeout without receiving the secure packet within the predefined time window.

Algorithm 1: *ReConv* Secure Communication

Offline: exchange S, *SD*;
Offline: decide *PRNG*;
Offline: choose $\{C_i\} \in S$;
Transmitter:
$S\prime$ = S;
repeat
  normal transmission;
    send normal packets;
    enable secure mode after P;
  secure transmission;
    $S\prime = S\prime + 1$;
    r = *PRNG(S\prime)*;
    $\{C_r\}$ = *SEL(S, r)*;
    SET(K, $g0$, $g1$, $g2$, $g3 \in \{C_r\}$};
    send secure packet;
    enable normal mode;
  normal transmission;
    switch to normal protocol;
until *transmission end*;
Receiver:
repeat
  normal transmission;
    receive normal packets;
    enable secure mode after P;
  secure transmission;
    use *SD* for the first time switch;
    $S\prime = S\prime + 1$;
    r = *PRNG(S\prime)*;
    $\{C_r\}$ = *SEL(S, r)*;
    SET(K, $g0$, $g1$, $g2$, $g3 \in \{C_r\}$};
    receive secure packet or timeout;
    enable normal mode;
  normal transmission;
    switch to normal protocol;
until *transmission end*;

3.2 Extended ReConv

ReConv is extended on two folds to increase the hacking complexity: byte level *ReConv* and fake CRC.

Byte Level ReConv

Each byte of a secure packet can be applied a unique set of convolutional parameters, since the proposed parameter switching at the proposed convolutional encoder and Viterbi decoder can be completed within a single clock cycle through multiplexer switching. This byte level parameter switching ability is named as *BL ReConv*. The size of the security key would be increased linearly; however, the hacking complexity

would be increased exponentially. For example, when two sets of convolution parameters are applied, the size of polynomial array G is doubled, while the attacker's brute force complexity would be squared. The choice of how many sets of convolution parameters are to be used for a single secure packet, is to be decided by the user after considering the trade-off between the security key size and the hacking complexity.

Fake CRC

A fake CRC mechanism is proposed to remove the easy secure packet identification. Once the *ReConv* secure communication mechanism is released to public, a smart eavesdropper may identify the secure packets from other packets with CRC error, especially when the wireless channel condition is good. A fake CRC is generated at the transmitter and appended at the end of the packet. When eavesdropper decodes the secure packet with the convolution parameter defined by Wi-Fi, the CRC comparison can be successfully passed by comparing the locally generated CRC at eavesdropper and the fake CRC. To achieve this ability, the signal processing at the transmitter would have to be increased greatly. The fake CRC generation mechanism is illustrated in Figure 3.3.

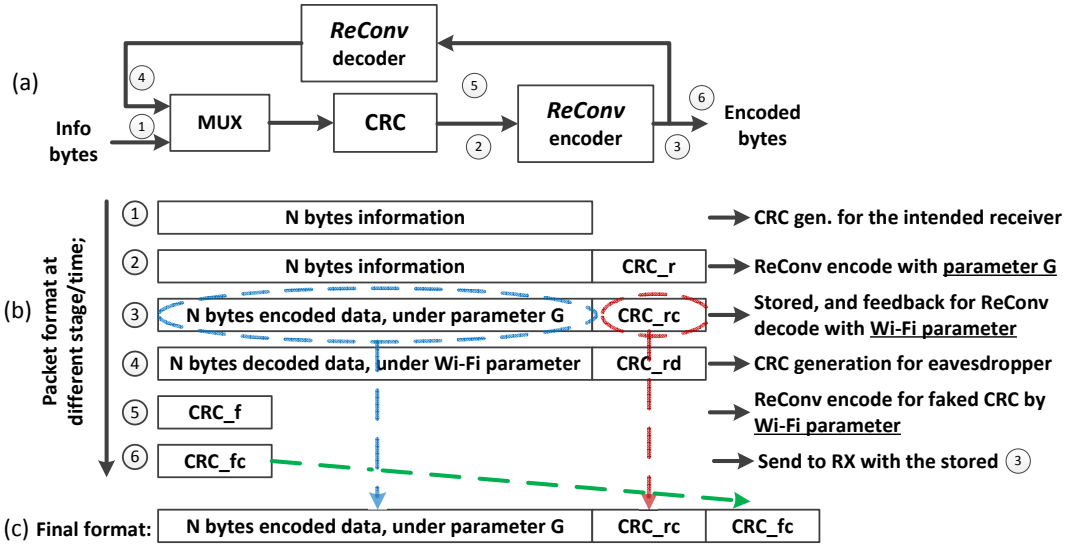

Figure 3.3: Fake CRC generation

The CRC for the intended receiver, CRC_r, is encoded to CRC_rc with convolution parameter G. The encoded data and CRC_rc are stored, meanwhile, they are fed back to the transmitter side through a Viterbi decoder which is fixed to the Wi-Fi convolution parameter. Both data and CRC can be decoded, however, due to the convolution parameter mismatch, the decoded data and CRC_rd are different from the original information data and CRC_r. In fact, these decoded results simulate the eavesdropper's behavior. Thus, the fake CRC, CRC_f, can be generated according to these decoded results. Finally the CRC_f is encoded to CRC_fc by the Wi-Fi convolution parameter and sent to the receiver with the stored encoded data and CRC_rc.

There are two CRC fields in the final encoded packet, shown in Figure 3.3 (c). The CRC_fc is for the eavesdropper and it will be discarded at the intended receiver. The

CRC_rc is for the intended receiver for error checking.

Figure 3.4 gives an example of interleaved transmission of a secure image and an unsecure image. When the fake CRC is not applied, the secure packets would be automatically dropped at the eavesdropper, thus the secure image cannot be sensed at the eavesdropper. However, a smart eavesdropper may enquire the reason behind the CRC errors. When the fake CRC is applied, the eavesdropper is also able to receive and open the secure image correctly; however, the image contents would look like random noise. Therefore, the eavesdropper may think this is the correct receiving image, and further investigation at the eavesdropper side can be stopped.

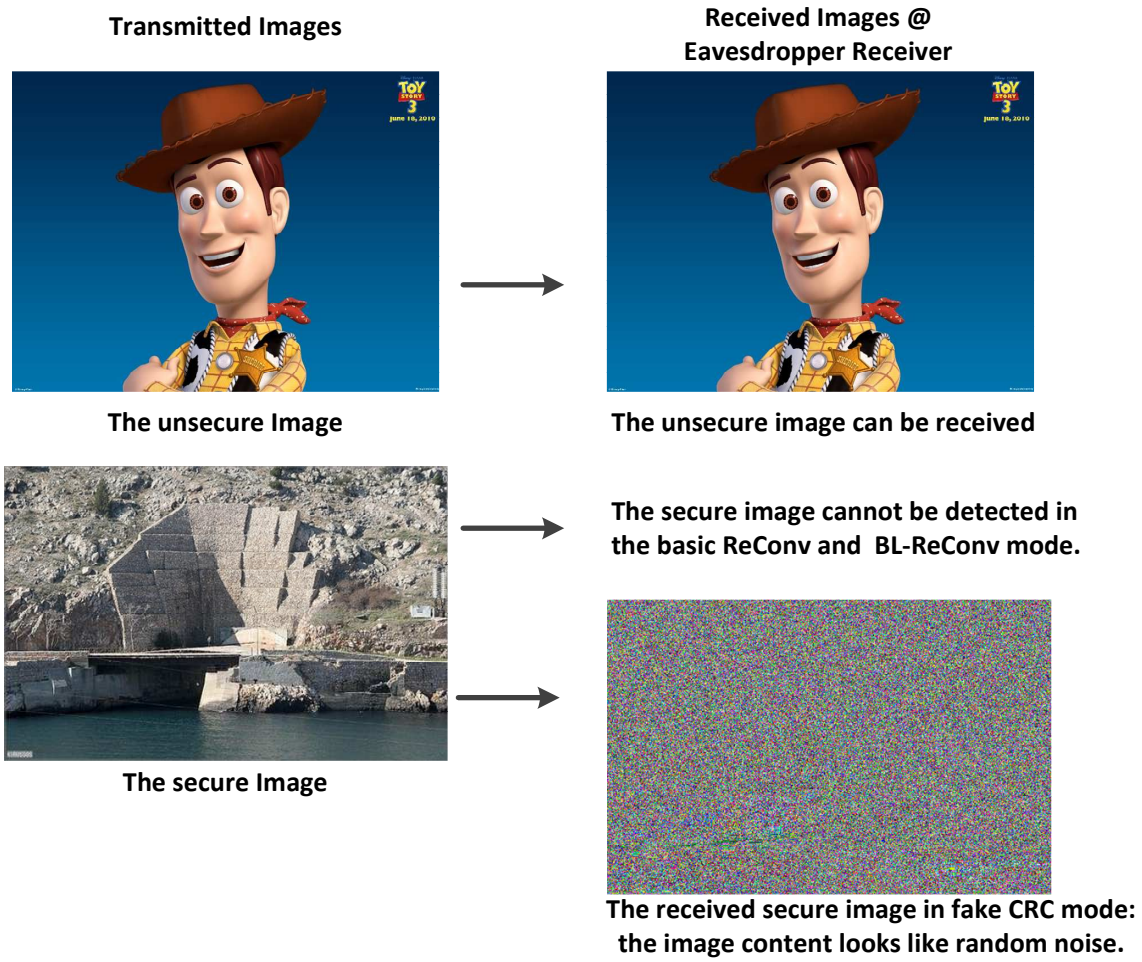

Figure 3.4: Secure image attacking scenario

3.3 ReConv Security Complexity

The secret key for *ReConv* is defined in Equation 3.1. Apart from the timing parameter P, the four polynomials array G defines the alternated convolution behavior. These polynomials create a large set of combinations for the secret key C_i. Each polynomial has 2^K combinations. Excluding the non-realistic "all zero" and "single one" (such as

the binary format: 000000010, the information bits will be sent directly through the XOR components) combinations, the number of applicable combinations for convolution, NUM_comb, is shown in Equation 3.2. The K is the constraint length, and B is the number of output branches. The maximum K and B are 9 and 4 in *ReConv*, and the NUM_comb is 63.5 billion.

$$NUM_{comb} = (2^K - (K+1))^B \quad (3.2)$$

$$complexity = NUM_{comb} * cycle\_VTB \quad (3.3)$$

$$cycle\_VTB = NUM_{bits} * cycle\_VTB/bit \quad (3.4)$$

The complexity for the eavesdropper to *predict* a secret key is extremely complicated, even if the eavesdropper could read the raw data. The complexity of attacking a *ReConv* secure packet is given in Equation 3.3. The $cycle\_VTB$ is the number of clock cycles to decode a packet in Viterbi decoding. The complexity is decided by the number of bits in the secure packet (NUM_{bits}), and the number of clock cycles to decode a single bit in Viterbi decoding ($cycle\_VTB/bit$). We explored the Wi-Fi $cycle\_VTB/bit$ on an Xtensa ASIP [23] processor. The profiling result from Xtensa shows that 14720 clock cycles are needed to decode a single bit. The eavesdropper will not be able to confirm the correctness of the predicted polynomials until the whole packet is decoded and the CRC operation is performed. Given a 1000 bytes packet, about 118 million (14720 * 1000 * 8) processor cycles are needed to estimate one key. Given a 2GHz processor, 59 ms would be consumed for a single guess of the key. To attack this secure packet, 119 years would be used with the Xtensa ASIP, if the 63.5 billion combinations of convolution polynomial parameter G have to be guessed. If an eavesdropper uses a hardware based Viterbi decoder for hacking, the $cycle\_VTB/bit$ can be reduced significantly. Ten clock cycles are needed to decode a single bit when our proposed reconfigurable Viterbi decoder architecture is utilised. For the 1000 bytes secure packet example, 96000 clocking cycles are needed to try one key. With 63.5 billion parameter combinations, an eavesdropper needs 35.3 days to attack the key by brute force with a 2GHz working clock.

When *ReConv* is working in the extended modes, it is more harder for an eavesdropper. When the *BL ReConv* is applied, the attacking complexity is increased exponentially. The eavesdropper has to try the all combinations of packet byte segmentations since he has no idea on the packet byte boundary for the different convolution parameters, and the complexity in Equation 3.3 applies to each segmentation. To successfully hack a secure packet, the convolution parameters from all segmentations should be matched with the packet sender's definition.

The fake CRC improves the security by another fold. Since the eavesdropper can pass the CRC check, he cannot identify which packet is the secure packet, and which packet is the normal packet. As the secure image example in Figure 3.4 shows, the transmitted secure image can be opened correctly. It is not easy to decide that the content of the image is meaningless.

We show a brief analysis on the security of *ReConv* in terms of information security. The underlying theory is creating an encoded data through customisation for security, which is only decodable by the intended receiver. Equation 3.5 shows the formulation for unicity distance from the information theory, where H(S) is the entropy of the key and D is the redundancy of the transmitted packet. Since there are NUM_comb possible variations to the key S and the packets (or contents in the packets) are independent to each other (i.e., equally probable to create the same encoded data), the unicity distance

of *ReConv* is U $\approx (log(Num_{comb})/0) = \infty$. This indicates that the attacker will not be able to determine whether the decoding is correct regardless of the number of encoded packets received.

$$U = H(S)/D \tag{3.5}$$

3.4 Reconfigurable Convolutional Encoder

Convolutional encoder encodes the information symbols with a series of concatenated shift flip-flops with additional XOR components. A serial implementation of the Wi-Fi convolutional encoder is shown in Figure 3.1. However, parallel implementation of convolutional encoder has become the common practice in the industry to improve throughput and reduce power consumption. A parallel implementation of the Wi-Fi convolutional encoder is shown in Figure 3.5 (a).

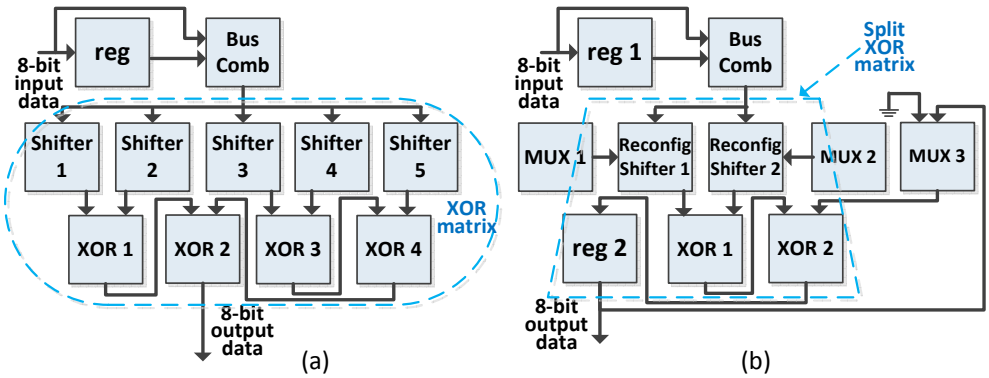

Figure 3.5: Reconfigurable convolutional encoder: (a) parallel implementation, (b) *ReConv* Encoder

We investigated the structure of parallel implementations of convolutional encoders and found that the XORs and shifters form a unique *XOR matrix* for each protocol. And the inputs for each tier of XOR operations in the *XOR matrix* are merely a shifted version of the buffered input bytes. Thus, this *XOR matrix* can be split into multiple stages such that the reconfigurable design can be efficient.

The proposed *ReConv* encoder is shown in Figure 3.5 (b). Same as the parallel encoder implementation, the input buffer register *reg 1* and *Bus Comb* construct a 16 bits input bus, including the current 8 bits input data and the previous clock cycle 8 bits input data. The two reconfigurable shifters, implemented by the barrel shifter architecture, two XOR components and *reg 2* form the split version of the *XOR matrix* in Figure 3.5 (a). The two reconfigurable shifters along with one XOR can perform the first XOR operation with two shifted versions of input data, and *XOR 2* performs the second XOR operation with the result of the first XOR operation and the buffered XOR result from the previous clock cycle. *MUX 1* and *MUX 2* select the correct configurations for the reconfigurable barrel shifters, while *MUX 3* can disable the feedback path from *reg 2* in the first iteration in the split XOR matrix. For the Wi-Fi convolutional encoder given in Figure 3.5 (a), three iterations are needed to complete the XOR matrix operation.

3.5 Parallel Viterbi Decoder Implementation

The reconfigurable Viterbi Decoder, i.e., *ReConv* decoder, is extended from the parallel Viterbi decoder (PVD) implementation which is introduced in this section. PVD can decode 8 states in parallel style and provide the high throughput Viterbi decoding. Its throughput is fast enough to support all the data rates in Wi-Fi.

Viterbi Decoding Algorithm

Trellis decoding is used for Viterbi decoding, which finds the most likely path in a trellis diagram, showing the state transitions of the convolution LFSR over time. The Figure 3.6 gives an example trellis diagram for a K=4 convolutional encoding. The eight cells in each column represent the all eight possible states for the convolutional LFSR. There are five columns for stage i to state i+4. Each stage associates to a decoding bit. The solid paths are the possible state transition when the input of the convolution LFSR is 0, while the dashed paths are the possible state transition when the input of the convolution LFSR is 1. The red lines indicate the most likely path, which reveals the convolution LFSR input bit sequence, i.e., the information bits before the convolutional encoding. The PVD's job is to find the most likely path to decode the convolutional codes.

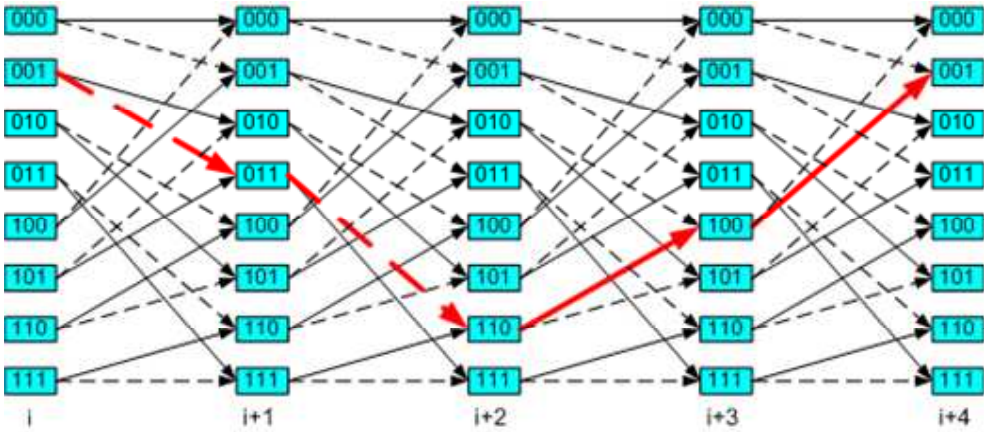

Figure 3.6: Viterbi decoding trellis diagram

The state i of stage k in a trellis diagram, as illustrated in Figure 3.7, has two possible states ($Z(0,i)$ and $Z(1,i)$) in the previous stage k-1, depending on the input of the LFSR (either "0" or "1"). Both branch metrics from state $Z(0,i)$ at stage k-1 and state $Z(1,i)$ at stage k-1, $\lambda_k^{(0,i)}$ and $\lambda_k^{(1,i)}$ respectively, are needed. Each branch metric is calculated by the individual branch at first, then all the branch results are summed together in to a single branch metric $\lambda_k^{(m,i)}$.

Both the received symbol and ideal symbol are required to calculate the branch metric for each branch. The ideal symbol is determined by the content of the LFSR and XOR polynomials. A local convolutional encoder, identical to the convolutional encoder in the transmission circuit, is needed to generate this ideal symbol. The output at each branch for this ideal symbol is only a single bit, which will be multiplied with the received symbol to measure the correlation.

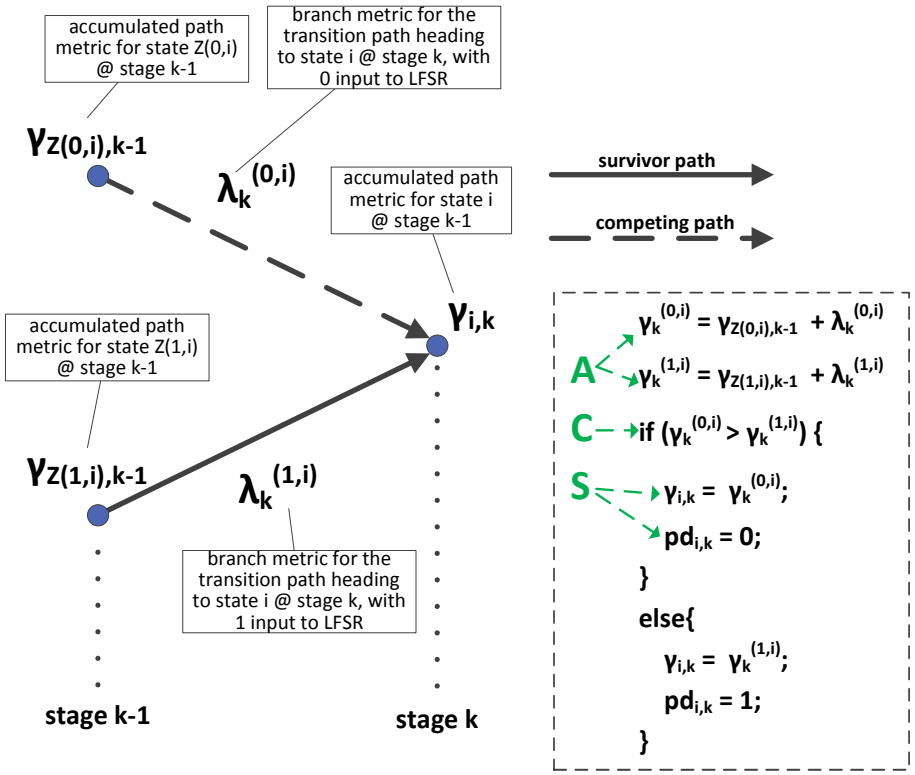

Figure 3.7: Survivor path ACS operation

The results of all branches are summed together to generate the final branch metric. For example, for 802.11a two branches Viterbi decoder, the inputs for state $Z(0,i)$ at stage k-1 are 0.9 for branch A and -1.2 for branch B. If the ideal branch A and B results are 1 and -1, then the branch metrics for $Z(0,i)$ at stage k-1 would be 2.1 = $0.9 \times 1 + (-1.2) \times (-1)$.

In the present context of Viterbi decoding, whenever two paths merge in one state, only the most likely path (the best path or the survivor path) needs to be retained. The path which is currently better will always stay better, within all possible extensions to the path [24]. This process is referred to as the add-compare-select (ACS) recursion. The path with the best path metric leading to every state is determined recursively for every step in the trellis. The metrics for the survivor paths for state i at stage k are named stage metrics $\gamma_{i,k}$. $\gamma_{i,k}$ is the best path metric for the paths leading to state i, which is the sum of the state metrics of the predecessor states and the corresponding branch metrics. A path metric is a sum of metrics of all branches in the path. The path metric of state i at stage k for m LFSR input $\gamma_k^{(m,i)}$ is defined in Equation 3.6:

$$\gamma_k^{(m,i)} = \gamma_{Z(m,i),k-1} + \lambda_k^{(m,i)}, \qquad m \in [0,1] \tag{3.6}$$

The $Z(m,i)$ is the state transition function which determines the predecessor state: $x_{k-1} = Z(m,i)$. The m is the estimated LFSR input bit and it can only be 0 or 1.

The state metrics $\gamma_{i,k}$ is determined by selecting the best path (survivor path):

$$\gamma_{i,k} \quad = MAX\left\{\gamma_k^{(0,i)}, \gamma_k^{(1,i)}\right\} \tag{3.7}$$

Figure 3.8 shows an example of survivor path selection. For the stage i at stage k, the following operations are executed: Firstly, the branch metrics $\lambda_k^{(0,i)}$ and $\lambda_k^{(1,i)}$ are calculated by measuring the distance between the received symbol y_k and the ideal channel symbol c_k. Secondly, the stage metrics on the previous stage (k-1) $\gamma_{Z(0,i),k-1}$ and $\gamma_{Z(1,i),k-1}$ are retrieved from memory. Thirdly, the path metrics $\gamma_k^{(0,i)}$ and $\gamma_k^{(1,i)}$ are calculated by adding the stage metrics on the predecessor states and the branch metrics. Lastly, By comparing the $\gamma_k^{(0,i)}$ and $\gamma_k^{(1,i)}$, the state metrics $\gamma_{i,k}$=$\gamma_k^{(1,i)}$ and the decision bit d_i, k 1 are determined.The $\gamma_k^{(1,i)}$ is the maximum path metric in state i at stage k. Both the state metrics $\gamma_{i,k}$ and decision bit d_i, k are stored in memory.

These ACS operations should be executed for each state in each stage. For a 802.11a (K=7, and 64 states) frame with 4096 bits, these ACS operations need to be iteratively executed 262144=4096*64 times. Working on 802.11a's highest data rate, i.e., 54Mbps, these 262144 iterative ACS operations must be finished within 4856μs [8], or 54 ACS operations per microsecond.

Despite the recursive computation, there are still N best paths pursued at the last stage. In communication protocols, this state of the last stage is predefined, called a terminated trellis diagram, so that the whole state transition path can be retrieved from the last stage according to the stored decision bits d_i, k.

The decoding latency and requirement on storage unit are proportional to the length of the trellis, especially when the packet frame is long. Furthermore, in applications like broadcasting, a continuous sequence of information bits has to be decoded rather than a terminated sequence. Fortunately, the acquisition and truncation properties of the Viterbi algorithm [24] allow the Viterbi algorithm to decode data when the final stage state is unknown, with negligible performance losses.

There are N ($2^{(K-1)}$) survivor paths at stage k on a certain trellis diagram. These paths merge, when traced back over time , called *survivor depth* D, into a single path. One example of the merged path is shown as the red line in Figure 3.8. This merged path is referred to as the *final survivor*. For trellis steps smaller than $k - D$, the paths will merge into the final survivor with very high probability. Thus, the trellis steps smaller than $k - D$ have a very high probability of being correctly decoded. Moreover, the requirement of storage memory can be reduced when the whole trellis is chopped into multiple decoding sections. The trellis steps from $k - D$ to k can be left to the beginning of the next decoding block so that these trellis steps can still be correctly decoded. The survivor depth D is usually selected as $D = 5K$ [25].

PVD Architecture

The block diagram of PVD is shown in Figure 3.9. The major components in PVD are branch metric unit (BMU) and add-compare-select (ACS), and traceback (TB).

The BMU and ACS complete the branch metric calculation, path metric accumulation and path decision, as detailed in the previous section. The TB generates the final output bits from the PD RAM, which is maintained by the ACS. Two accumulated metrics storage (AM) RAMs (13*64 bits each) are used to store the accumulated path metrics and two path decision storage (PD) RAMs (96*64 bits each) are used to store the path decision bits for each decoding truncation. The CTRL module is the state machine manager for the whole PVD.

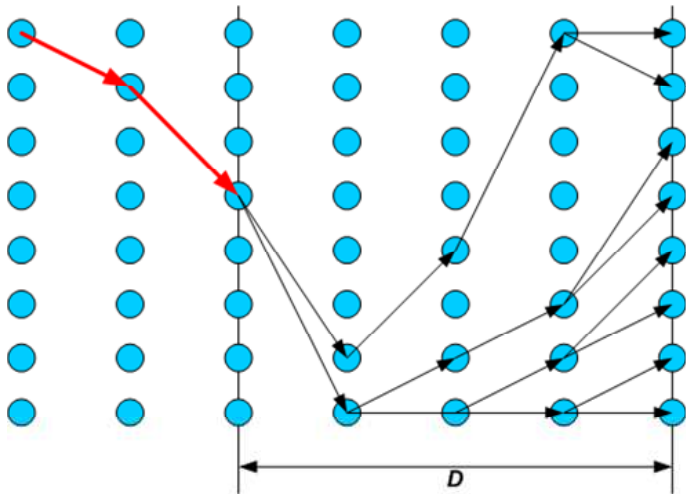

Figure 3.8: Final survivor merging after survivor depth D

The signal *sta_acs* feeds the starting state for each parallel ACS operation. BMU generates the ideal convolution LFSR output based on the state number specified by *sta_acs*, and the branch metrics are produced by the correlation of this ideal LFSR output and the received coded symbol. ACS accesses AM/PD RAM to update the accumulated path metrics and path decision, according to the branch metrics from BMU and the previous accumulated path metrics. The *stage_done* assertion signal, utilised to control overflow, indicates ACS that all the states in a stage are processed. Traceback is initiated using the *tb_start* signal. *sta_tb* signal gives the state number of the starting stage of traceback.

The input bitstream is chopped in to multiple overlapping truncations to save the RAM usage [26]. The length of each truncation is fixed to 96 bits in PVD, and the last truncation can be shorter.

Parallel processing of states is applied in PVD to accelerate the BMU and ACS operations. There are 2^{K-1} states on each stage, i.e., there are 2^{K-1} operations to decode a single bit. For Wi-Fi, there are 64 states since the K is 7. PVD can process eight states on each cycle for BMU and ACS respectively. Thus eight cycles ($64/8 = 8$) are needed for a single stage processing in Wi-Fi Viterbi decoding, and 768 cycles ($96 \times 8 = 768$) are needed for the 96-stage truncation decoding. However, due to the truncation characteristic [26], the last 32 bits may not be decoded correctly and they need to be decoded again in the next truncation. Both BMU and ACS have the same parallelism structure. BMU generates the branch metrics for eight states within one cycle and feeds to ACS in a pipeline manner. The ACS then updates the path metrics for the assigned eight states in the next cycle. The TB processing is relatively simple, which traces the each decision bits in one cycle. 96 cycles are required to trace a truncation since the truncation length is 96.

ACS needs to update the path metrics by reading and writing to the AM RAMs. To avoid the AM RAM read/write conflicts, two AM RAMs are utilised: a reading AM RAM holds the accumulated path metrics for the previous stage, a writing AM RAM stores the calculated path metrics for the current stage. When all the states in the current stage are processed by ACS operations, the role of the two AM RAMs are

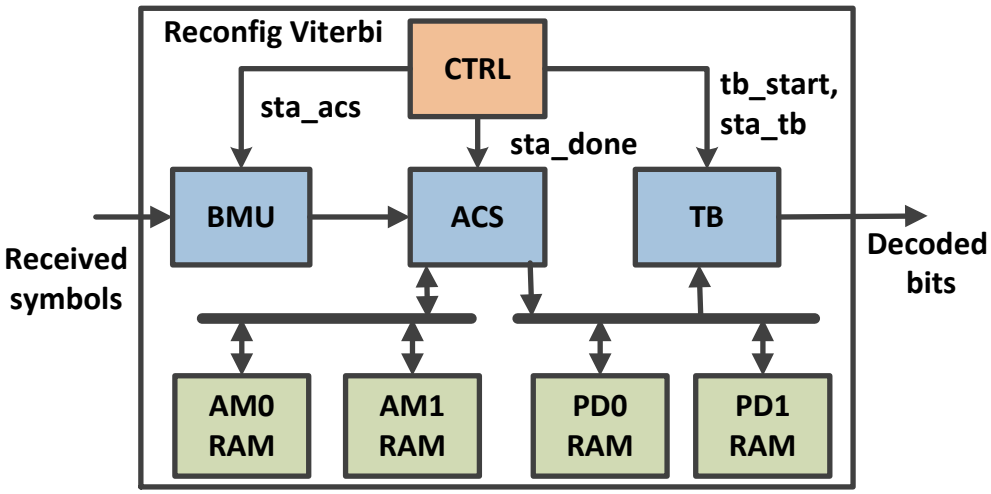

Figure 3.9: Parallel Viterbi decoder block diagram

swapped (i.e., the read AM RAM will become the write AM RAM and the write AM RAM to the read AM RAM). Similarly, two PD RAMs are used to avoid the read/write conflicts. When one PD RAM is written by ACS, the other PD RAM will be read by TB to traceback the decoded bits. When ACS finished its current truncation operation, the role of the PD RAMs will be swapped.

3.6 Reconfigurable Viterbi Decoder

Viterbi decoder is the decoding function block for the convolutional encoded data. Due to the highly complex signal processing capabilities required for Viterbi decoding, a hardware implementation is preferred.

A hardware based reconfigurable parallel Viterbi decoder, RPVD, is proposed based on our 8-state parallel implementation, which is detailed in section 3.5. The block diagram of the parallel Viterbi implementation is shown in Figure 3.9. Only two modules, branch metric unit (BMU) and add-compare-select (ACS), are updated from the parallel Viterbi implementation for the reconfigurable Viterbi decoder, which are detailed in the following sections.

Reconfigurable BMU

BMU is used to calculate the branch metric which measures the correlation between the noise included received symbol and the locally generated *ideal* symbol. The stronger correlation indicates the received signal is more likely to match with the ideal signal. The linear distance is utilised in RPVD as the branch metric, since it is less complicated and shows good performance [27].

The reconfigurable BMU supports different combinations of polynomial set G , constraint length K and coding rate R.

Both the received symbol and ideal symbol are required to calculate the branch metric for each branch. The ideal symbol is determined by the content of the LFSR and XOR polynomials. A local convolutional encoder, identical to the convolutional encoder in the transmission circuit, is needed to generate this ideal symbol. The output

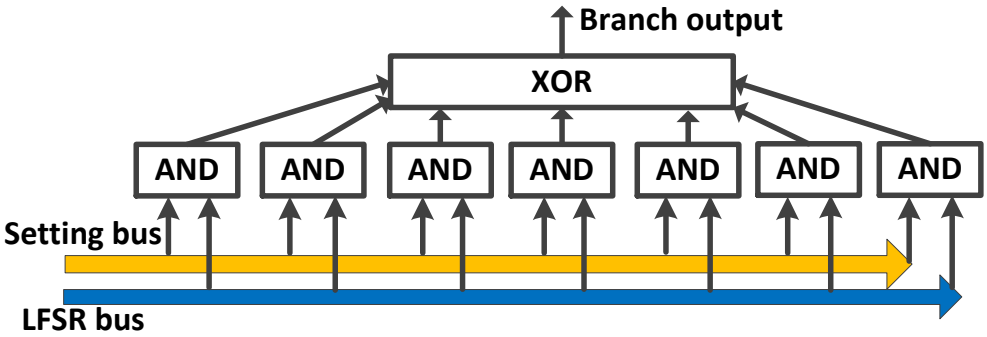

Figure 3.10: Reconfigurable convolutional output branch generation

at each branch for this ideal symbol is only a single bit, which will be multiplied with the received symbol to measure the correlation. Since the lowest coding rate R is four in *ReConv* , each serial convolutional encoder has four output branches. The key part of reconfigurable BMU is the processing of reconfigurable XOR operation polynomials g_0 - g_3, which defines the behaviour of the serial convolutional encoder. The block diagram of one reconfigurable XOR branch is illustrated in Figure 3.10. The *LFSR bus*, including LFSR input and content, is fed to nine AND gates. These AND gates are enabled by the *setting bus*, which selects the inputs for XOR operator from the *LFSR bus*. By changing the *setting bus*, different XOR operations can be completed.

The number of states at each decoding stage is the major factor for the Viterbi decoding throughput as each state needs a branch metric calculation. The number of states for a given constraint length K can be defined as: 2^{K-1}. For example, Wi-Fi has 64 states with K=7. An 8-state parallel processing technique is applied in the parallel Viterbi decoding, thus, 8 iterations are needed to calculate the 64 branch metrics for a single bit decoding in Wi-Fi, and each iteration only consumes one cycle. When the number of states is changed due to a change in constraint length K, only the number of iterations will be changed.

The convolution coding rate affects the error correction capability. The number of output branches equals to $1/R$, and all the output branches results are summed up to provide a single branch metric for the specific state. Although the branch outputs summation circuit is fixed, the output of the unused branches would be zero as the corresponding polynomial parameters are zero. For example, the polynomial G: $g_0 = 133, g_1 = 171, g_2 = 0, g_3 = 0$, defines the two output branches used for Wi-Fi convolutional coding.

Reconfigurable ACS

ACS updates the path metrics and decides the path decision bit for the specific state in the specific stage, the detail ACS concept and parallel implementation can be found in section 3.5.

Only the constraint length K affects the reconfigurable ACS, and the polynomials G and coding rate R have no impact on the ACS operation. Similar to the reconfigurable BMU, several iterations are needed to process all the states for a single bit decoding and the number of iterations is reconfigurable by the *CTRL* module.

The accumulated path metrics and the path decision need be stored in the AM and PD RAMs, as described in section 3.5. A larger value of K needs larger AM and PD

RAMs. Sixty four element entries are needed for both the AM and PD RAMs. The size of each entry in the AM/PD RAM is given in the section 3.5.

4 EXPERIMENTAL SETUP

ReConv has been implemented in Matlab and RTL for security protocol verification and hardware performance evaluation.

The whole Wi-Fi baseband is designed in Matlab at first. Then the convolutional encoder and Viterbi decoder are replaced by the *ReConv* encoder and decoder. The *ReConv*, *BL ReConv* and fake CRC modes are verified.

ReConv, including the reconfigurable convolutional encoder, Viterbi decoder and CRC, is designed in RTL. To compare the hardware performance overheads, a standard Wi-Fi convolutional codec is also implemented. Both *ReConv* and Wi-Fi RTL have the same Viterbi decoding settings: 7-bit soft decision, truncation length 96 and 8-state parallel processing. *ReConv* is forced to support the same constraint length as the Wi-Fi for fair comparisons, i.e., K=7. All eight data rates in Wi-Fi 802.11a, listed in Table 4.1, are supported by the proposed *ReConv*.

Table 4.1: Wi-Fi 802.11a data rates

	R0	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**
Data rate (Mbps)	6	9	12	18	24	36	48	54

The test vectors for *ReConv* and Wi-Fi RTL are first generated using the Matlab program. These test vectors are then fed into both the RTL testbenches. RTL simulations are performed (using Mentor Graphics ModelSim) and the functionality of these implementations is verified before performance analysis. Area and timing results are generated by Synopsys Design Compiler with TSMC 65nm technology, and power consumption is estimated by Synopsys Prime Time. Same experimental settings (such as frequency, input/output delay, etc.) are applied to both *ReConv* and Wi-Fi RTL.

5 RESULTS and ANALYSIS

5.1 Secure Protocol Verification

The *ReConv* integrated Wi-Fi baseband Matlab program transmits an unsecure photo and a secure photo (Figure 3.4) in the interleave style as defined in Algorithm 1.

ReConv has been set to the basic *ReConv*, *BL ReConv* and fake CRC modes. The intended receiver can receive and decode both photos correctly. The simulated eavesdropper cannot receive the secure photo in the basic *ReConv* and *BL ReConv* mode, due to the CRC errors on the all packets for the secure photo. In the fake CRC mode, the secure photo can be received by the eavesdropper, however, the content is just random noise, as shown in Figure 3.4.

5.2 Hardware performance analysis

Hardware area and throughput

The area results of *ReConv* and Wi-Fi implementations are summarized in the second row of Table 5.1, whereas the throughputs in Mega bits per second (Mbps) at

the transmission (TX) and receiver (RX) are provided in the third row and forth row respectively.

It is worth noting that the RAM memories in *ReConv* and Wi-Fi are implemented as register banks since we do not have a proper RAM model. About 90% gates are consumed by RAM memories in both *ReConv* and Wi-Fi implementations.

The gate count of *ReConv* is 177.6K, which is only 5% higher than Wi-Fi (168.6K).

Table 5.1: Area and throughput comparison

	ReConv	**Wi-Fi**
Gate count (K)	177.6	168.6
Throughput (TX) (Mbps)	2694	8889
Throughput (RX) (Mbps)	84	93

The throughput is determined by the maximum working frequency, and the number of processing cycles per symbol decoding. The maximum working frequency is determined by decreasing the clock period in the synthesis constraint file until timing violation is occurred.

We show how the throughput is calculated using an example. The maximum working frequency of *ReConv* is 1010 MHz. Eight cycles will be used to decode a single bit in the *ReConv* Viterbi decoder. For every 96 bits decoding, only 64 bits are considered valid while the other 32 decoded bits will be discarded. Thus, 12 cycles are required to decode a single bit hence the *ReConv* throughput (at transmission) is about 84Mbps (84=1010/12). Similarly, the throughput of Wi-Fi implementations are: 8889 Mbps/TX, 92 Mbps/RX.

The *ReConv* throughput in transmission is much slower than Wi-Fi implementation. This is due to the iterative reconfigurable convolutional encoder occupying more clock cycles for encoding. While in receiving mode, the difference in throughput between *ReConv* and Wi-Fi is only about 10%, due to the longer critical path delay in *ReConv*.

Power consumption analysis

Power consumption is a critical factor in mobile devices. *ReConv* is power efficient due to two reasons. First, the logic size of *ReConv* is only 5% more than Wi-Fi convolutional codec. Hence the overhead in terms of static power consumption is minimal. Second, the parallel processing of both convolutional encoder and decoder is supported. Thus, the working frequency of *ReConv* can be kept low while complying with the data rate listed in the specification, thereby reducing the dynamic power. Figure 5.1 shows the power consumption comparisons between *ReConv* and Wi-Fi implementations. Clock gating is applied to both implementations for TX and RX (to reduce the dynamic power consumption when the circuits are not used).

ReConv consumes 1.15/32.6 mW for TX/RX in average. Compared to the 1.09/31.4 mW average for TX/RX in Wi-Fi, *ReConv* consumes 5% more power in TX and 4% more in RX.

Communication performance evaluation

The secure packet is transmitted in *ReConv* with the convolutional encoding parameters which are different with the Wi-Fi specification. To explore the impact on the

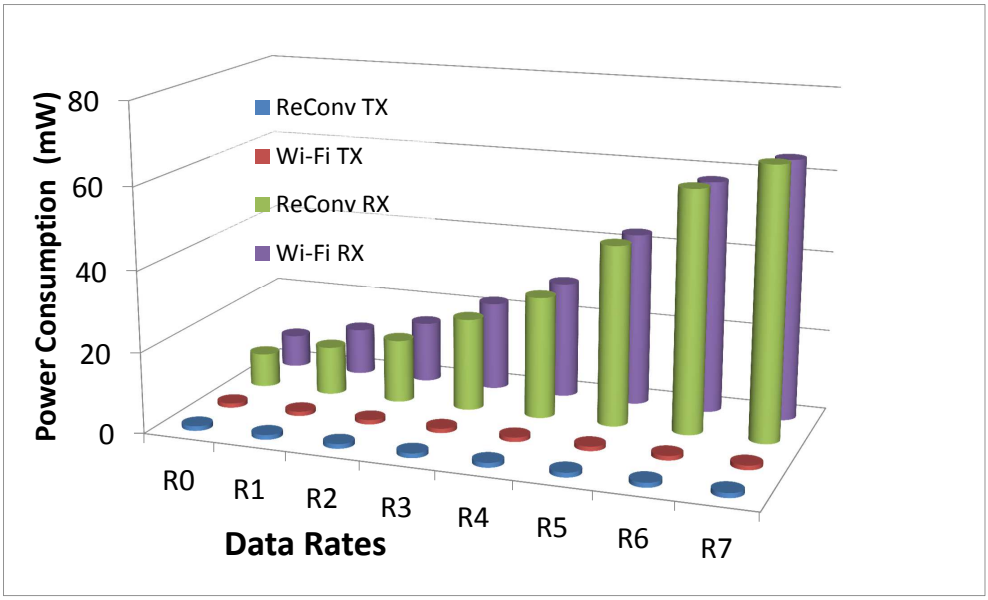

Figure 5.1: Power consumption comparison

communication performance by these parameters which are not from Wi-Fi specification, the packet error rate (PER) is evaluated and the results show the communication performance degradation is very small.

The whole Wi-Fi 802.11a baseband is implemented in Matlab, then noise is injected into the AWGN channel. The measured PER is plotted in Figure 5.2. The y-axis presents the packet error rate (PER) for different Signal-to-Noise-Ratios (SNR) in dB (shown in x-axis). Due to space restrictions, we report 15 possible convolution parameter settings (out of the possible 63.5 billion sets of parameters), as listed in Table 5.2. The test case names are listed in ID column, whereas their coding rate and constraint length are listed in R and K columns respectively. The g0, g1 and g2 columns reveal the polynomial definitions for each test case. The contents ”0” of polynomials mean the corresponding convolutional output branches are disabled. For example, A7_0 has a coding rate of 2, hence only the output branch 0 and 1 are valid. The g3 for all test cases are 0.

Table 5.2: PER test case parameter setting

ID	R	K	g0	g1	g2		ID	R	K	g0	g1	g2
A7_0	1/2	7	133	171	0		U7_0	1/3	7	133	171	165
A7_1	1/2	7	163	135	0		U7_1	1/3	7	163	135	165
A6_0	1/2	6	132	162	0		U5_0	1/3	5	114	144	124
A5_0	1/2	5	114	144	0		U4_0	1/3	4	130	150	70
A4_0	1/2	4	130	150	0		U3_0	1/3	3	120	60	120
A3_0	1/2	3	120	60	0		U2_0	1/3	2	140	140	40
A2_0	1/2	2	140	140	0		S7_0	1	7	133	0	0
							S6_0	1	6	132	0	0

A7_0 utilises the parameters from Wi-Fi specification and provided as the baseline.

U7_0 uses the parameters from UWB specification [28]. A7_1 only differs with the A7_0 in polynomial definition, and the PER of A7_1 and A7_0 are nearly identical as shown in Figure 5.2. For the other test cases which has a constraint length (K) less than 7 and coding rate is 1/2, PER is worse than A7_0. However, even for the constraint length of 2 (i.e., A2_0), the PER is reachable to 0% when SNR is 7.5 dB. Test cases which have R=3 outperform the cases with R=2. This is due to the additional branch output at the convolutional encoder when R=3. It is worth noting that case U6_0 is not shown because it overlaps the A7_0 baseline curve. The R=1 cases reach 0 PER when SNR nears 9.5dB. Figure 5.2 indicates the different combination of convolution parameters can protect the secure packet against channel noise and interference, albeit the error correction ability can be worsen in some cases, comparing to the convolution codec defined by the specification. The worst communication PER from the single branch convolutional encoding, S6_0, is about 7.7dB lower than the Wi-Fi PER.

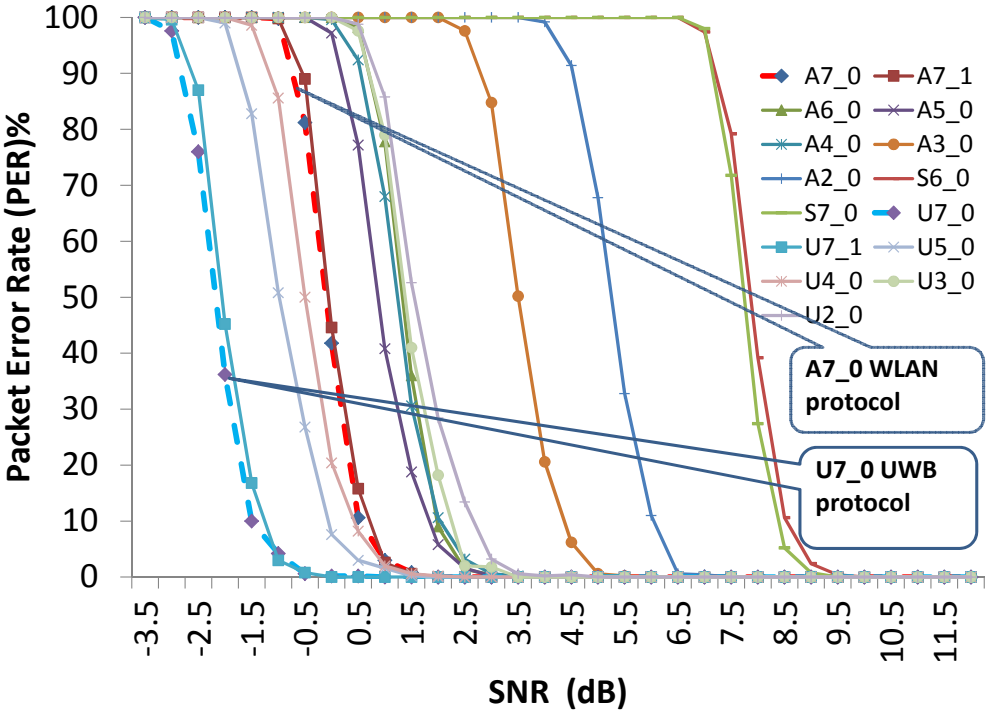

Figure 5.2: PER simulation results

5.3 Discussion

ReConv provides a secure communication mode by switching between the standard convolutional codec and alternated convolutional codec. The switching period of *ReConv* is only a single cycle so that the byte level parameter setting is achieveable.

The convolutional encoding parameters are changed in *ReConv* for the secure packet transmission. We explored the communication performance by these alternated parameters. The detail results can be found in section 5.2.

Wi-Fi convolutional encoder is implemented and compared with *ReConv* in this paper. Any other protocols whose constraint length is not greater than 9 can be supported

by our *ReConv*. The support of the longer constraint length can be easily implemented by increasing the RAM size and number of iterations in the *ReConv* decoder CTRL module. Thus *ReConv* is a good candidate for low cost, high performance software defined radio (SDR) convolutional codec.

Apart from the convolutional code, Turbo code is also used in many protocols for error correction, such as LTE. We are extending *ReConv* to reconfigurable Turbo code in the future.

6 CONCLUSION

In this paper, a physical layer secure communication is proposed by a fast switching, low overhead, high performance reconfigurable convolutional encoder and decoder. We state that this is a low cost ultra secure methodology. The secure packet, encoded by the alternated convolutional encoder, will not be successfully decoded by an eavesdropper at the physical layer.

Bibliography

[1] A. Lindell, "Test Attacks on the Pairing Protocol of Bluetooth v2.1," in *BlackHat*, 2008.

[2] B. Yu and H. Li, "Research and design of one key agreement scheme in bluetooth.," in *CSSE (3)*, pp. 665–668, IEEE Computer Society, 2008.

[3] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Lecture Notes in Computer Science*, 1999.

[4] J. Quisquater and D. Samyde, "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards.," in *E-smart*, pp. 200–210, 2001.

[5] "Bluetooth specification version 1.2," 2003. Available at: http://www.bluetooth.org/.

[6] J. Cho, Y. Kim, and K. Cheun, "A novel frequency-hopping spread-spectrum multiple-access network using m-ary orthogonal walsh sequence keying," *IEEE Trans. on Computers*, vol. 51, no. 11, pp. 1885–1896, 2003.

[7] T. Kang, X. Li, C. Yu, and J. Kim, "A survey of security mechanisms with direct sequence spread spectrum signals," *Journal of Computing Science and Engineering*, vol. 7, Sep 2013.

[8] "Wireless lan medium access control (mac) and physical layer (phy) specifications," 2011. Available at: http://standards.ieee.org/.

[9] M. I. Husain, S. Mahant, and R. Sridhar, "CD-PHY: Physical Layer Security in Wireless Networks through Constellation Diversity," *CoRR*, vol. abs/1108.5148, 2011.

[10] M. Nutzinger, C. Fabian, and M. Marschalek, "Secure Hybrid Spread Spectrum System for Steganography in Auditive Media," in *IIH-MSP*, pp. 78–81, 2010.

[11] Q. Mao and C. Qin, "A novel turbo-based encryption scheme using dynamic puncture mechanism.," *JNW*, vol. 7, no. 2, pp. 236–242, 2012.

[12] E.-K. Lee, M. Gerla, and S. Oh, "Physical layer security in wireless smart grid," *Communications Magazine, IEEE*, vol. 50, no. 8, pp. 46–52, 2012.

[13] Q. Ling and T. Li, "Modeling And Detection Of Hostile Jamming In Spread Spectrum Systems," in *SAFE*, pp. 1–5, 2007.

[14] A. Martin, Y. Hasan, and R. Buehrer, "Physical layer security of hybrid spread spectrum systems," in *RWS*, pp. 370–372, 2013.

[15] A. Chorti and H. Poor, "Faster than Nyquist interference assisted secret communication for OFDM systems," in *ASILOMAR*, pp. 183–187, 2011.

[16] A. Chorti, "Helping interferer physical layer security strategies for M-QAM and M-PSK systems," in *CISS*, pp. 1–6, 2012.

[17] M. Daly and J. Bernhard, "Directional Modulation Technique for Phased Arrays," *IEEE Trans. on Antennas and Propagation*, vol. 57, no. 9, pp. 2633–2640, 2009.

[18] A. Polydoros and C. Weber, "Optimal Detection Considerations for Low Probability of Intercept," in *MILCOM*, vol. 1, pp. 2.1–1–2.1–5, 1982.

[19] T. Rappaport, *Wireless Communications: Principles and Practice*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2nd ed., 2001.

[20] S. A. Cohen, "Interference Effects of Pseudo-Random Frequency-Hopping Signals," *IEEE Trans. on Aerospace and Electronic Systems*, vol. AES-7, no. 2, pp. 279–287, 1971.

[21] W. Kummer, A. Villeneuve, T. Fong, and F. Terrio, "Ultra-low sidelobes from time-modulated arrays," *IEEE Trans. on Antennas and Propagation*, vol. 11, no. 6, pp. 633–639, 1963.

[22] A. Babakhani, D. Rutledge, and A. Hajimiri, "A Near-Field Modulation Technique Using Antenna Reflector Switching," in *ISSCC*, pp. 188–605, 2008.

[23] "Xtensa Processor." Tensilica Inc. (http://www.tensilica.com).

[24] "Viterbi Decoders: High Performance Algorithms and Architectures." Available: http://www.eecs.berkeley.edu/newton/Classes/EE290sp99/lectures/ee290aSp996_1/vit_chap17.pdf/.

[25] J. G. C. Clark and J. B. Cain, *Error-correction coding for digital communications*, vol. Second Printing. New York: Plenum Press, 2. print. ed., August 1982.

[26] R. Cypher and C. B. Shung, "Generalized trace-back techniques for survivor memory management in the viterbi algorithm.," *VLSI Signal Processing*, vol. 5, no. 1, pp. 85–94, 1993.

[27] H.-L. Lou, "Linear distances as branch metrics for viterbi decoding of trellis codes," in *Proceedings of the Acoustics, Speech, and Signal Processing, 2000. on IEEE International Conference - Volume 06*, ICASSP '00, (Washington, DC, USA), pp. 3267–3270, IEEE Computer Society, 2000.

[28] “Multiband ofdm physical layer specification,” 2005. Available at: http://www.wimedia.org/.