Identity Management for Interoperable PTC Systems in Bandwidth-Limited Environments

The Final Report

Part 3 (of three parts)

The Proposed Solution

Prepared for the Federal Railroad Administration
Under Grant Number FR-TEC-0006-11-01-00/20.321

Version prepared on July 26th, 2014 as a revision to the version from May 2014.

Principal Investigator: Prof. Rajni Goel, Howard University

Co-Principal Investigators: Prof. Duminda Wijesekera, George Mason University

Dr. Andre B. Bondi, Siemens Corporation

Table of Contents

Executive Summary 3

Section 1. Hash Seed Generation Methodology 4

1.A. Objectives and Requirements 4

1.B Hash Generation Principal 4

1.B.1 Lamport Schema 5

1.B.2. TESLA 5

Section 2. Hash Generation Protocols for WIUs 7

2.1. Distribute Cryptographic Parameters 8

1. The back-office sets the cryptographic parameters 8

2. Provide Cryptographic Parameters to Trains Admitted into the Region 9

3. On Receipt of Cryptographic Material, WIUs or TRAINS Compute Hash Sequences 9

4. WIU Computes Hash value and Transmits a Broadcast Packet 9

5. A Train Receives a Packet and Checks Integrity (perfect Synchronization). 10

6. A Train Receives a Packet and Checks Integrity (imperfect Synchronization). 11

Section 2. Implementing the WIU Message Hash Generation 14

WIU Client and Server Composition 15

Development Branch Decomposition 16

Use of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) 18

Representation Invariant 18

References 20

List of Figures

Figure 1. 1: TESLA [Perrig2002] 6

Figure 2. 1: Cryptographic Engine 7

Figure 2. 2: Time Based Communication Diagram 12

Figure 2. 3: Phased Development 14

Figure 2. 4: Server and Client Process Call Graphs 16

Figure 2. 5: Software Branch Timeline 17

Figure 2. 6: FRA-3 Branch – Command Shell 18

List of Tables

Table 2. 1: Describing Modules in Figure 2.1 8

Table 2. 2: Explanation of Figure 2.2 12

Table 2. 3: Explanation of the Developmental Phases Shown in Figure 2.4 15

Table 2. 4: Software Roadmap 17

Executive Summary

Positive Train Control is a wireless based system designed to provide comprehensive safety coverage for passenger and cargo trains operating on US railroads by 2015. Mandated by Rail Safety Improvement Act of 2008 (RISA 2008), major railroads have designed a broad architecture consisting of two networks; namely the Signaling Network (SN) and the Wayside Interface Network (WIN) powered by software-defined radios (SDRs) that use the same 220MHz range. The Signaling Network provides authorities for trains to enter fixed blocks of track and other signal functions and the Wayside Interface Network provide sensory information about the vicinity of the tracks. The railroad community has decided that both network require message integrity and availability but not confidentiality for both networks.

From published documents, the Wayside Interface Network uses truncated SHA-1 hashed keys to ensure the integrity of the WIU messages. We have found that this choice may weaken the security requirements of WIU message broadcasts. We demonstrate these vulnerabilities using the details of the proposed protocols.

Part 2 of this report showed that the existing wayside interface protocol has vulnerabilities. This part (Part 3) describes a solution that overcomes those vulnerabilities. Our solution to overcome the hash breaking attack is to use a different hash for every hash at the every time moment. This way the beacon's integrity values will not be repeated over for a long time. Given that to change hashes frequently require precise clocks, we first show a solution that operates under this strict assumption and show a relaxed version that does not depend on precisely synchronized clocks.

Although we have listed all investigators, the work reported in the section was solely done by the George Mason University team consisting of Anthony Melaragno, Damindra Bandara and Duminda Wijesekera.

Section 1. Hash Seed Generation Methodology

This section describes our way to avoid the vulnerabilities shown to exist in the WIU beaconing protocol.

1.A. Objectives and Requirements

The basic objective is to create a new hash key per every message beacon generated by any given WIU and to ensure that that two different WIU's do not generate the same beacon. The method described in this report assumes that the WIU network is reachable by the back office on a periodic basis.

In order to formally specify, let there be WIU's $\{W_i: 1 \le i \le n\}$ be the collection of WIU's. Let $\{S_{i,j}: 1 \le i \le n, 1 \le j\}$ be the jth hash seed of the ith WIU. Then the requirement to have all hash seeds of all beacons to be distinct is formally specified as $\forall i,j,i',j'\ [(i,j) \neq (i,',j') \Rightarrow S_{ij} \neq S_{i'j'}]$

1.B Hash Generation Principal

Our method for every WIU to generate a different seed uses the two networks. First when a train is granted entry to a segment (such as a block or a consecutive set of blocks) the back office uses the signaling network to hand over three parameters. For the sake of explanation, assume that these parameters are two different primes and a time that starts (or started) a hash generation period. Say that the two primes are p and q and t. These values are also conveyed to the WIU's by the back office.

Then after j time units (say j seconds) ith WIU uses the jth hash seed created by using a TESLA like protocol that started with a generating seed p^i. In addition, at the (t+j)th time WIU i uses a different hashing algorithm. Suppose there are M hash functions approved to be used for hash generation and f(a,b,c,d) be any one-to-one (injective) function. Then we can use [f(p,q,i,j) mod m] to be the function used to create the hash by the ith WIU at time t+j. An example of f(a,b,c,d) is $2^a3^b5^c7^d$. Due to the unique prime factorization of any number, $[2^a3^b5^c7^d = 2^{a'}3^{b'}5^{c'}7^{d'} \Rightarrow a=a' \wedge b=b' \wedge c=c' \wedge d=d']$ thereby showing that f is a one to one function. Consequently, the (t+j)th beacon message uses hash seed p^iq^j and uses the hashing algorithm [f(p,q,i,j) mod m].

Given that changing the hash per every message require that both clocks at the WIU beacon and the recipient On Board Unit (OBU) be synchronized exactly. But we are aware that at best both clocks can be synchronized against a GPU clock. Consequently, we allow a time period, say $\Delta(i)$, a validity period for the i^{th} (hash, algorithm) pair. Our system is based on an enhancement of the TESLA (Timed Efficient Stream Loss-tolerant Authentication) [Perrig2005] protocol. TESLA was based on a previously developed one-time password schema by Lamport [Lamport1981]. In order for this report to be self-contained, we first, describe the necessary details of Lamport's schema and the TELSA protocol.

1.B.1 Lamport Schema

Lamport's schema has two steps.

In the first step, the schema starts with

1. An initial seed (say X) and
2. A seeded hash function (say H) like SHA-256 and
3. N, the number of different hash keys desired to create hash values for data.

The second step creates a hash key chain of length with h_1=hash(X,H) and h_{n+1}=hash(h_n,H). Then the required sequence of hash keys are [h_{n+1}, h_n,….,h_1]

The security of the schema uses the fact that creating h_i from h_{i+1} involves reversing a hash, because h_{i+1}=hash(h_i,H) and relies on the length of the hash key and the strength of the hashing algorithm H.

1.B.2. TESLA

The TESLA protocol uses the same idea as Lamport's, scheme, but associates every has value with a time interval (that is the time between receiving two successive has values) Figure 1.1 shows example time intervals of using a 4-item long hash chain. This will require that the system and the client be time synchronized. Limiting the time during which a password is valid reduces the risk of a replay attack. In the case of OBU/WIU communication, changing the HMAC key at previously agreed times reduce the risk of a replay attack or hash spoofing, provided the original generation seed is not exposed to an imposture.

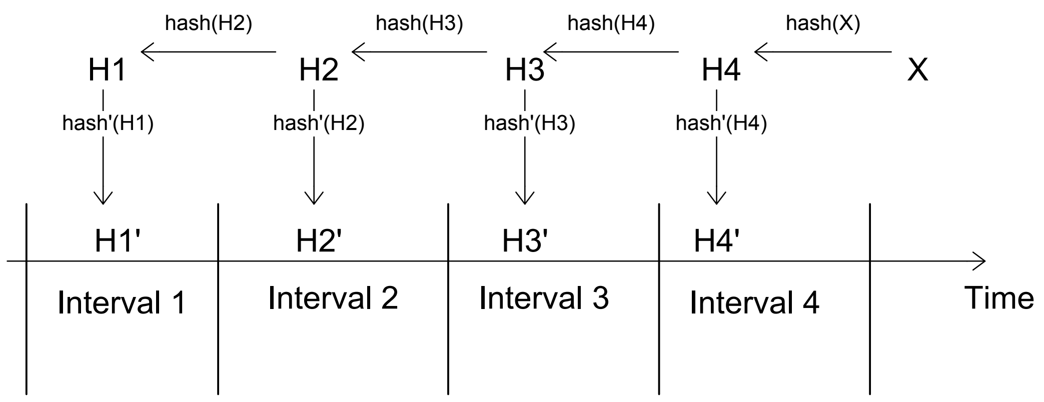


Figure 1. 1: TESLA [Perrig2002]

We now describe our enhancement of the TESLA protocol for secure WIU broadcasts. Conceptually, in order to use TESLA for secure WIU broadcasts, we need to address issues that are not addressed in the original TESLA protocol or any of its enhancements (to the best of our knowledge).

1. Reaching an agreement on the length of a hash chain and other parameters.
2. Finding an alternate solution in case the hash chain finishes prior to establishing a new one.
3. The protocol used by a train that enters a region that already has a running hash chain.
4. Time synchronization failures to determine the time interval of hash validity.

Now we summarize the conceptual framework we have created up to now for solving these issues.

1. We use the signaling network in using agreement on the proposed cryptographic parameters, including the key lengths.
2. We have developed and implemented a fallback solution in case key chains run out.
3. We describe a protocol used by a train that enters a region that has an already running hash chain.
4. Currently we use a preset time synchronization window of 30 seconds for hash key interval, but have not found any synchronization problems. Ongoing work experiments are attempting to obtain variable time synchronization windows in a WIU network.

Section 2. Hash Generation Protocols for WIUs

This section describes protocols to generate secure hashes for the WIU network. The following diagram segments the processes and associated data messages necessary to support the architecture. The figure illustrated below describes a cryptographic engine which processes received WIU messages and validates the message, the CRC, the Integrity Value, and the logical consistency.

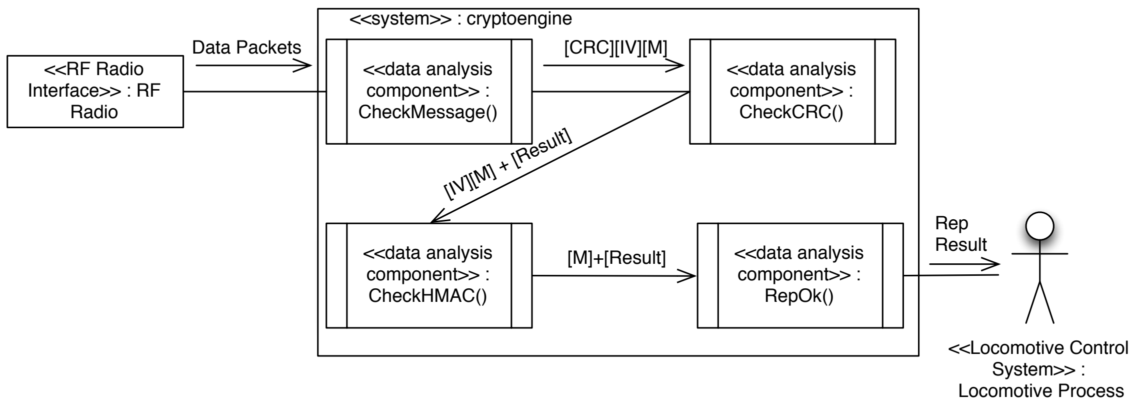

<<system>> : cryptoengine
Data Packets
<<RF Radio Interface>> : RF Radio
<<data analysis component>> : CheckMessage()
[CRC][IV][M]
<<data analysis component>> : CheckCRC()
[IV][M] + [Result]
<<data analysis component>> : CheckHMAC()
[M]+[Result]
<<data analysis component>> : RepOk()
Rep Result
<<Locomotive Control System>> : Locomotive Process

Figure 2. 1: Cryptographic Engine

The system can be decomposed into functional components, which are used to determine the integrity and logical consistency of a message, as shown in Figure 2.1. Table 2.1 describes the functional contributions of each module in Figure 2.1. The receiver component of the RF interface is connected to the crypto engine and accessible through a standard Ethernet interface. Data packets are received and converted to a readable format. Conceptually the following occurs and will be described in greater detail in the following section:

- The message is checked for the correct data length prior to processing
- The message is queued and then checked for data corruption during transmission (CRC Errors)
- The CRC is stripped off the packet and the result of the CRC check is queued
- The IV is checked to ensure that the data message has not been tampered with and the result is checked
- The data representation invariant is checked as well as the results from the previous steps

Component	**Message Interface**	**Description**
RF Radio	RF Radio Interface	The (RF) radio sends data packets (that encapsulates the WIU message) using TCP or UDP. Once de-capsulated, the WIU Message is placed into a queue.
Check Message	Data Packets	Queued WIU messages are dequeued and checked for data integrity and sent to the checkCRC function. This is symbolized as [CRC][IV][M].
Check CRC	WIU Message Datagram	The WIU message CRC is checked and the result placed into a searchable database for forensic analysis. If successful the HMAC is checked.
Check HMAC	WIU Message Datagram	The WIU Message is checked against the pseudo randomly selected (salt, algorithm) pair for a chosen time period. The result is placed into a database for later forensic analysis.
Rep Ok	WIU Message Datagram	Once verified, the representation invariant is checked for logical consistency. The CRC hash value, the message, time period, and expected state of the system are compared and action is taken as designed.

Table 2. 1: Describing Modules in Figure 2.1.

As mentioned in the table, all items are logged so that forensics can be analyzed concurrently while the radios are communicating and post communication event. This would allow for near real time situational awareness and action as the analysis of the data is communicated back to the signaling network. The architecture is designed for future expansion and research to support a cognitive engine to evaluate risk conditions. We now describe the details of the designed and implemented system.

2.1. Distribute Cryptographic Parameters

On a regular time interval (say the i^{th} time interval since the start of timing in the WIU network) the protocol SetCryptoParameters(_) is issued by the back office to set the cryptographic parameters of the j^{th} time interval.

1. The back-office sets the cryptographic parameters

SetCryptoParameters(Start of Time: t_0, Time Interval: Δ, number of hash functions: M, hash sequence length: N)

{Back Office: $(t_0,\Delta,M,N) \rightarrow$ WIU Network

Back Office: $(t_0,\Delta,M,N) \rightarrow$ Signaling Network

```
    }
```

2. Provide Cryptographic Parameters to Trains Admitted into the Region

GiveCryptoPrametesToTrains(Start of Time: t_0, Time Interval: Δ, number of hash functions: WIU Message m, hash sequence length: N)

```
    {Time of granting passage of Authority
    Signaling Node($t_0$,Δ,m,n) → OBU
    }
```

3. On Receipt of Cryptographic Material, WIUs or TRAINS Compute Hash Sequences

On receipt of cryptographic parameters (t_0,Δ,M,N) the WIU creates the hashes to be used on broadcast messages.

CreateTESLAhashes(Start of Time: t_0, Time Interval: Δ, number of hash functions: M, hash sequence length: N)

```
    {
    CryptoQueue keyAlgorithmChain

    //$p^i$ represents the specific prime used for a particular broadcast area
    // the prime acts as the initial seed for salt creation
    HashArray[0]:==$p^i$;
    for(j:0;j≤N)
        AbstractDataType cryptoelement;
         cryptoelement.crypto = CreateHash(HashArray[$p^i$])
        cryptoelement.algorithm = HashAlgorithm($p^I$ mod M);
        keyAlgorithmChain.push(cryptoelement);
}
```

4. WIU Computes Hash value and Transmits a Broadcast Packet

WIUComputingHash(Message wiuMessage, Start of Time: t_0, Time Interval: Δ, number of hash functions: M, hash sequence length: N)

```
    {x: time
```

```
x::=getTimeOfDay();
char hash[32];
if ($t_0$<x< $t_0$+,$\Delta$.n){                  /* within n time intevals */
        i::= $\lfloor$(x- $t_0$)/ $\Delta\rfloor$ ;              /*Computes the hash time interval*/
        hashKey::= HashArray[N-i];          /* Derive the hash key*/
        hashseed  = (keyAlgorithmChain.pop()).cryptoelement
        IV ::= hash (hashseed xor wiuMessage);      /*compute hash key*/
packet::=createpacket(message,IV, header, trailer);
transmit(packet);
}
```

5. A Train Receives a Packet and Checks Integrity (perfect Synchronization).

The following method to check integrity assumes that the WIUs and Trains have perfectly synchronized clocks. Next we present an enhancement that will relax these assumptions.

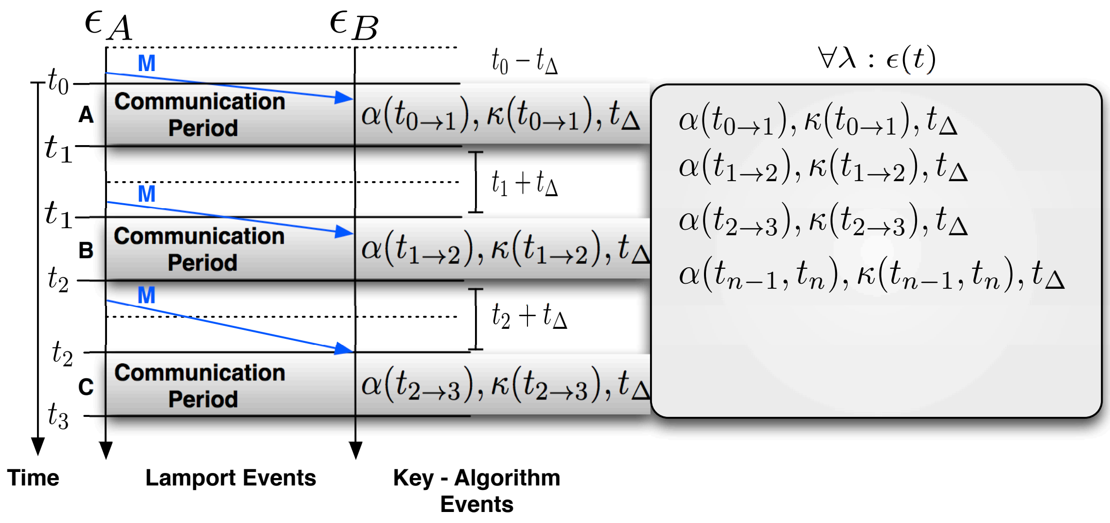
```
getPacket(packet: p);
    {       Integer: Interval,
            Start time: $t_0$,
            Time Interval: $\Delta$,
            Number of hash functions: M,
            Hash Sequence length: N,
            time: t
            //Testing if we got the entire valid message
            // Since we know the protocol length if we mod the protocol by the
            // protocol length we then know if it is valid or not – first check
            if(p mod messageSize == 0) // Valid Message
            {       // Extracting the first 32 bits from the received message
                    char *rx_crc = (char *) memncpy(p, 0, 32)
                    // Extracting the rest of the CRC from the packet
                    char *calc_crc = (char *) memncpy(p, 32, sizeof(p)
                    // We are now calculating the CRC for message errors
                    calc_crc = crc(calc_crc)
```

```
                if(strncpy(rx_crc, calc_crc, 32) == true)
                {       // Passed the CRC error checking
                        wiuMessage[sizeof(WIUmessage)]
                        //extracting the wiu message
                        memncpy(p+32, sizeof(wiumessageframe), wiuMessage)
                        if(WIUComputingHash (wiuMessage)== True)
                        {       Message is correct
                                Perform actions
                        j::= packet.wiuId;
                        h::=packet.hash;
                        t::=getTimeOfDay();
                        i::= $\lfloor(x- t_0)/ \Delta\rfloor$ ;                /∗ Derive the hash key∗/
                        hashKey::= HashArray[N-i] ;    /* compute hash key*/
                        hashAlgo::= hashKey (mod m);
                        hash::= createHash(p.content, hashAlgo,hashKey)
                        /*create message hash*/
                        }
                        else
                        {       Log the message
                        }
                }
                else
                {       // Log – Date, Time, Location CRC error and message
                        // Information for later forensic investigation
                }

}
```

6. A Train Receives a Packet and Checks Integrity (imperfect Synchronization).

When a packet is received by the locomotive's OBU two conditions need to be checked. The first is that the CRC needs to be properly verified to ensure that the packet was not corrupted in transmission. The second condition is that the integrity value (IV) has been verified. This is done traditionally by taking a hash of the message.



ϵA
ϵB
$t_0 - t_\Delta$
$\forall\lambda : \epsilon(t)$
M
A
Communication Period
$\alpha(t_{0\to1}), \kappa(t_{0\to1}), t_\Delta$
$t_1 + t_\Delta$
B
$\alpha(t_{1\to2}), \kappa(t_{1\to2}), t_\Delta$
$t_2 + t_\Delta$
C
$\alpha(t_{2\to3}), \kappa(t_{2\to3}), t_\Delta$
$\alpha(t_{n-1}, t_n), \kappa(t_{n-1}, t_n), t_\Delta$
t_0
t_1
t_2
t_3
Time
Lamport Events
Key - Algorithm Events

Figure 2. 2: Time Based Communication Diagram

The keys used in our hash checking algorithm is explained using Figure 2.2, where the vertical axis illustrates the flow of time, event, and the key in a set. The description of the variables and there purpose is shown in Table 2.3:

Variables	**Purpose**
$\alpha \in \alpha_{algorithms}$	A cryptographic algorithm is chosen from the set of all available hash functions for the time period t. The time period runs from t_0 to t_n.
$K_t \in K_{salts}$	A cryptographic salt is generated and used for the time period and represents a salt that is dequeued from the FILO queue. The methodology for generating and using the salt follows from the μTesla specification.
t_Δ	As in μTesla, the time difference encapsulates processing delay, propagation delay, clock drift, and an error factor. During the delta time period both the current and next key are considered *active*, and we check the received packet's integrity using both.
Lamport Event	Each message received with the correct IV for the time period would increment the lamport clock. The lamport clock would act as another synchronization point.

Table 2. 2: Explanation of Figure 2.2

When a message is received the OBU looks at the received time and the current time if it is within the time delta it uses the algorithm and key. If it is not it will begin increment to the next key, algorithm, for the appropriate time period. The following pseudo code can explain the interaction especially in an expiration time period.

Function Syncrhonizeit(IV)

```
{
    currentTime = getTime()
    // Current successful count of evnets
    eventCounter = getCounter()
    CryptoEntry entry = CryptoEntry SearchKeyChain(eventCounter)
    IVTime = entry.getIVTimePeriod
    If(IVTime  == currentTime)
        Syncrhonized
    Else
        If(IVTime < currentTime – (Propgation Delay + Processing Delay))
            eventCounter++
            return ((CryptoEntry) KeyChain.pop())
        if(IVTime > currentTime – (Propgation Delay + Processing Delay))
            eventCounter—
            return entry
    Endif
}
```

Section 2. Implementing the WIU Message Hash Genera

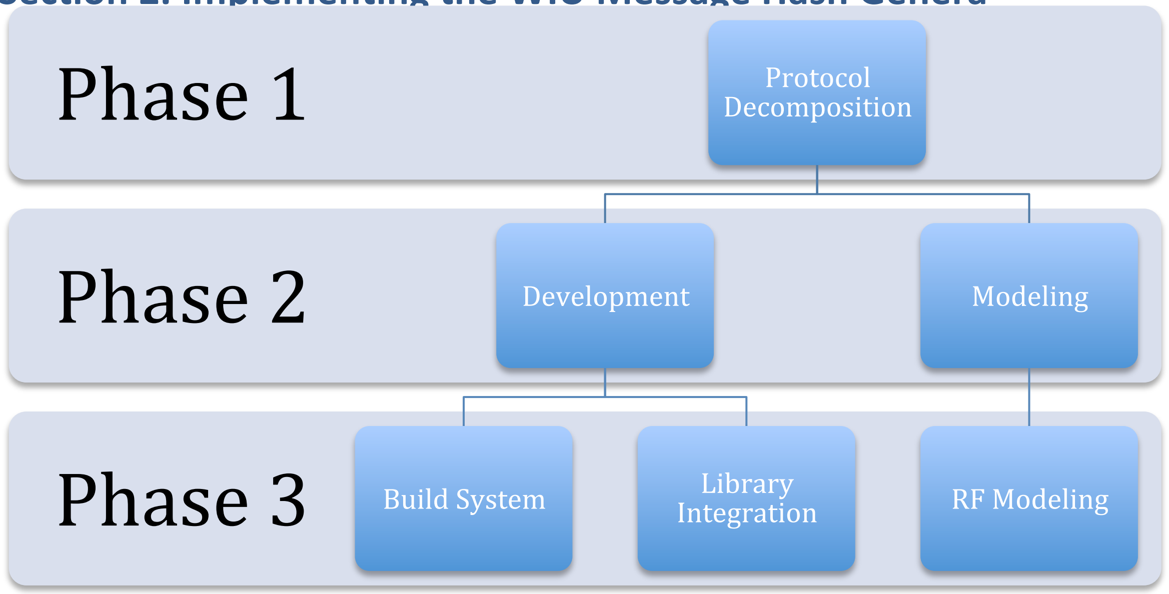


Figure 2. 3: Phased Development

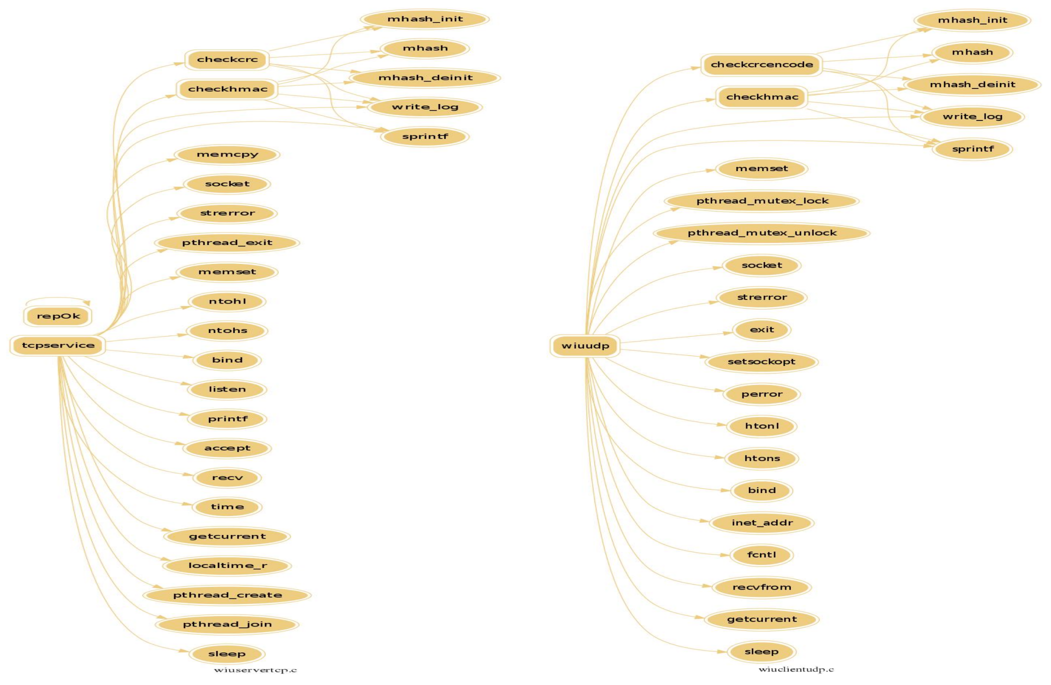
We implemented the software using open source cryptographic libraries, specifically, libmhash, openssl, libgcrypt, and linux queuing. The libraries provided proven implementation of the software of the critical cryptographic libraries. The phases that were used to develop the software is illustrated in Figure 2.4. Table 2.4 explains the activity done during each of the phases.

Phase	Phase Title
Phase 1: Protocol Decomposition	The purpose of the phase is to identify and decompose the protocol message set into implementable components.
Phase 2: Development and Modeling	The phase had two distinct goals and objectives: **Development**: Identifying Software, development and analyzing results. **Modeling:** Analyzing the Radio Frequency (RF) for issues pertaining to the communications environment
Phase 3: Component Development	**Build System:** Using Autoconf / autotools / libtools were to define the build environment for maximum portability across platforms. Additionally, Doxygen was used to document the developed libraries. **Library Integration:** Installed chosen libraries were installed in a Linux environment. Initially, Ubuntu 12.04LTS, as compatibility improved with GNU Radio, a decision was made to move to a Redhat variant therefore Fedora 20 was chosen. The cryptographic libraries are retrieved using the Fedora package manager. **RF Modeling:** The RF environment was modeled in Matlab to model multipath and RF obstructions for an urban and high-speed conditions.

Table 2. 3: Explanation of the Developmental Phases Shown in Figure 2.4

WIU Client and Server Composition

We developed WIU client and server architectures utilizing open source software. As previously mentioned Fedora Linux was used as an operating system as well as open source cryptographic libraries in phases 2 and 3. The physical layer was abstracted by using TCP and UDP protocol where the WIU data frames were embedded within the datagrams. CRC and IV values were evaluated though using the approach.



(a) Server Process (b) Client Process

Figure 2. 4: Server and Client Process Call Graphs

We show the server and client processes decompositions in Figure 2.5. Each of the processes is started by a calling *pthread*. Continuous salt generation and algorithm selection is not included in the first phase of development. Therefore all of the keys and algorithms follow Perrig's modified approach. It was later implemented as a proof of concept and awaiting integration into the main branch. As illustrated in the call graph check functions are used to check the HMAC for the hash for the intended time period and check CRC is used to check for data corruption.

Development Branch Decomposition

The development of the software began with developing a common library that is shared by both the client and the server through specific software branches. The software roadmap is described in table 2.5.

Branch Name	**Branch Description**	**Baseline No.**
FRA-1 cryptoengine → checkFuncions(): • CRC • HMAC	1. Defined and integrated core prototype functions 2. Integrated into RF Radio 3. Defined base RepInvariant Interface	FRA-1.0.2 (Status Completed)

• Base RepInvariant • Check Message		
FRA-2	Continuous Generation of cryptographic selection both the keys and algorithms.	FRA-2.0 (Status Completed)
FRA-3 Cryptoengine→SyncrhonizeIt()	Synchronization algorithm and prototype to synchronize communications.	FRA-3.0 (Status Completed)
FRA-4	Python test environment, which integrates the various branches in a test environment, which can be called and tested.	FRA-4.0 (Development)

Table 2. 4: Software Roadmap

As described in table 2.5, FRA branches were created to integrate into the main branch of FRA-4. FRA-4 allows for testing the WIU in an environment protocol under a variety of experiments and threat scenarios figure 2.6 illustrates the python environment that allows for future red team testing.

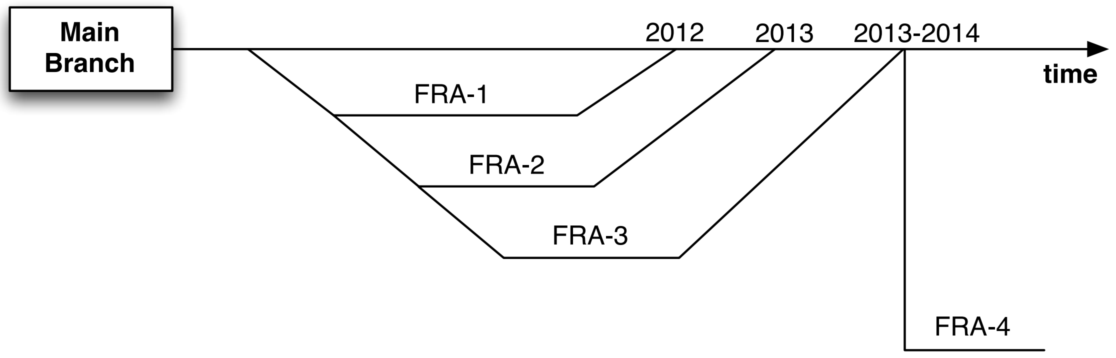
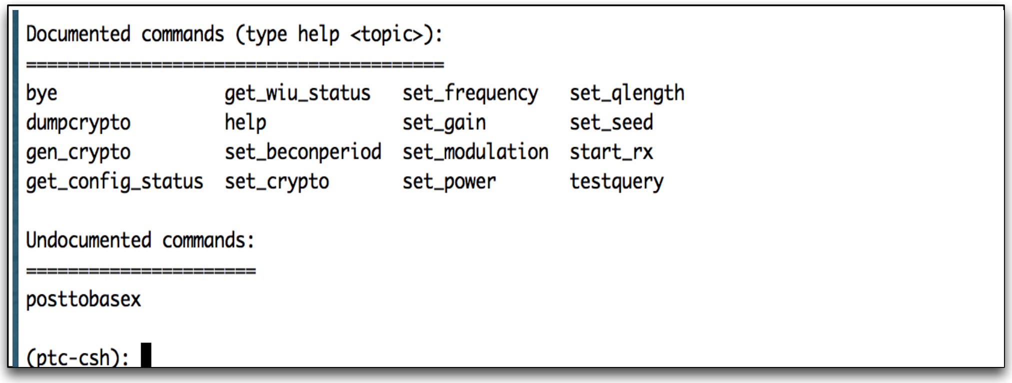


Figure 2. 5: Software Branch Timeline

Branches were completed and at FRA-4 the integration of the software into a python environment has started. FRA-4 acts as the final integration point to fully test the WIU protocol under a variety of security constraints in an RF environment and will allow integration of the Risk Engine.



```
Documented commands (type help <topic>):
========================================
bye               get_wiu_status   set_frequency   set_qlength
dumpcrypto        help             set_gain        set_seed
gen_crypto        set_beconperiod  set_modulation  start_rx
get_config_status set_crypto       set_power       testquery

Undocumented commands:
======================
posttobasex

(ptc-csh): 
```

Figure 2. 6: FRA-3 Branch – Command Shell

As illustrated in Figure 2.6 a python command shell was created to support a variety of security and WIU specific messages. This will allow greater interactions and experimentation to fully test the resilience of security methods including the Tesla integration.

Use of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP)

Two protocols were used in the initial design TCP and UDP. The layer 3 (IP) and 4 (TCP/UDP) protocols were chosen due to the simplicity of using a socket interface. The TCP/IP and UDP/IP interface was used to interface to the Network Interface Card (NIC) card that communicated to the radio. The request messages, such as get WIU status were encapsulated in TCP packets. Additionally, it had the CRC as well as the IV value. Once the transmission started the WIU packets were encapsulated into UDP packets. The project abstracted away the physical layer to make it possible to prove the cryptographic prototype. The next phase of the project would encompass physical aspects of the Radio Frequencies.

Representation Invariant

The representation invariant provides a logical check to verify that the data itself is consistent and does not pose a risk to the system. The following is an algorithm description of the Representation invariant.

```
Function ErrorCode RepInvariant(WiuMessage M, int WiuMessageType){
    If(M mod WiuMessageType != 0)
        ErrorCode = MessageError
    If(M.wiu_id != Possible WiuIds
                ErrorCode = PossibleWiuIds
return ErrorCode
}
```

The error code will then be used to verify characteristics of the communication environment and possibly provide post communications forensics as well as continuous situational awareness of the communications environment.

References

[Perrig2002] Perrig,A. and Tygar J.D., Secure Broadcast Communication in Wired and Wireless Networks , Kluwer Academic Publishers, 2002,101 Philip Drive, Assinippi Park, Norwell Massachusetts 020601 USA,1st,0-7923- 7650-1.

[S9001] S9001 ITC – System Reference Architecture, Version 1.4, Interoperable Train Control Architecture Team, August 2011.

[S9202] AAR Manual of Standards and Recommended Practices. Interoperable Train Control: Wayside Interface Unit Requirements, Standard S-9202, adopted 2012.

[S9352A] PTC Office-Locomotive Segment ICD, Release 2.10, 2/29/2012.

[S9352B] S-9352B Interoperable Train Control (ITC): Wayside-Locomotive Interface Control Document. DRAFT FOR COMMENT – DO NOT USE. Version DRAFT 3, 21 December 2010.

[S9352C] AAR Manual of Standards and Recommended Practices: Office Architecture and Railroad Electronics Messaging. S-9352C: ITC TIME AND LOCATION—INTERFACE CONTROL DOCUMENT (ICD), adopted 2012.

[Wang2005] Xiaoyun Wang, Yiqun Yin, and Hongbo Yu, Finding collisions in the full sha-1, In Proceedings of Crypto, August 2005

[WIU-AAR9202_2010] Interoperable Train Control Wayside Interface Unit Requirements Railway Electronics-AAR S-9202_2010