Secure and Seamless Payment for Wireless Mesh Networks

Serhat Can Leloğlu, Can Yücel, Albert Levi

Sabanci University, Turkey

canleloglu@sabanciuniv.edu

canyucel@sabanciuniv.edu

levi@sabanciuniv.edu

Abstract— Wireless Mesh Network (WMN) is a multi-hop high-speed networking technology for broadband network access. Compared to conventional network service providing systems such as base stations, WMNs are easy to deploy and cost-effective systems. In this paper we propose a secure and seamless pre-payment system for Internet access through WMNs. The proposed system is called and will be mentioned as SSPayWMN. The system will be fair to both clients and to service providers. Since service providers intentionally or unintentionally may overcharge the clients, SSPayWMN offers cryptographic proofs for given Internet service. Additionally SSPayWMN protects clients’ anonymity and provides unlinkability for the client actions. The implementation of the system is made on a network simulator and simulation results are presented in this paper. SSPayWMN has achieved remarkable results in the simulations; system protocols reached steady state in every simulation, which ensures the stability of the system.

Keywords—**Wireless** Mesh Networks, Cryptography, Payment Systems, Security, Network Simulation

1. Introduction

Wireless Mesh Networks [1] offer broadband network access with high-speed network connection. WMNs are easy to deploy and cost effective compared to conventional Internet service providing infrastructures such as high-powered servers. Mesh networks dynamically organize themselves and they do not need a centralized element, in that sense they are a subset of ad-hoc networks. Mesh nodes deliver packets from source to destination in a multi-hop manner, conclusively they extent network coverage.WMNs could support for both mesh purposes and also conventional Wi-Fi connections. WiMax [18], ZigBee [19] and 3G-radio access [20] could also inter-connect with WMN structure.

There has been research for developing secure pre-payment systems for Internet access. In [8], the authors use a high-level approach for billing and propose architecture. Their focus is mostly its performance on a threshold based bandwidth management algorithm. In [9], the authors propose UPASS; a double hash chain based prepaid billing architecture for WMNs. Their trust model is based on both classical certificate-based public-key cryptography and identity-based cryptography. The drawbacks of [8] are the complex trust and payment structures, missing simulative and/or analytical performance model, and disregarding users' anonymity/privacy. Similarly, UPASS does not consider client anonymity and unlinkability. The proposed secure and seamless system will implement a prepaid billing scheme with simpler structures and trust models. Authentication, user and operator non-repudiation, settlement and especially user privacy is taken into consideration in the system design.

SSPayWMN employs some cryptographic primitives to ensure system security. The billing system counts on hash chains [10] and uses every element of the hash chain as a token, which buys time intervals with Internet service. SSPayWMN employs a Trusted Third Party (TTP), who ensures honest usage of the system by every party. The packets that are transmitted are either encrypted or transmitted on a secure line.

SSPayWMN is designed to reckon with real-life challenges such as stable Internet service during client mobility and rush hours. To estimate SSPayWMN performance, network simulations for the proposed system are executed. The simulations are divided into two groups. The former is unit tests, which simulate a unit of the system and check if it is fit to use. A unit in SSPayWMN corresponds to network protocols. The latter simulation group is called real-life scenario simulations. In these simulations the clients are selected considering human behavior and they are grouped into different groups. Unit simulations provided considerable results and in all of the simulations SSPayWMN reached steady state performance. In real-life scenario simulation results the system reached steady state also, which ensures system stability.

The rest of the paper is organized as follows: First we give a brief overview for SSPayWMN and suggested network topology in Section 2. In Section 3 we explain the system protocols. The settlement of the operators and the money transfer is explained in Section 4. Simulation environment is explained in Section 5 and unit test results are presented in Section 6. Discussion on system properties takes place in Section 7. Finally conclusion is given in Section 8.

1. General Overview of Proposed Scheme and System Entities

The proposed system is a secure pre-payment infrastructure for WMNs that also considers users' privacy and fairness. In this infrastructure there are mobile phones or laptops as clients, as well as tools that are used for service providing. Table 1 gives a list of system entities that function in the proposed system.

TABLE I

System Entities

|  |  |
| --- | --- |
| C:\Users\Public\Pictures\client.png | Mobile user (client) |
| C:\Users\Public\Pictures\ap.png | Access Point (AP). From now on in this document, it is called as AP, but please note that it also has routing capability. |
| C:\Users\SUUSER\Documents\GitHub\worddoc\thesisImages\meshBackbone.png | Mesh backbone |
| C:\Users\Public\Pictures\gateway.png | Gateway that connects APs to Operators |
| C:\Users\Public\Pictures\operator.png | Operator's server (OP). Keeps necessary logs and user info. |
| C:\Users\SUUSER\Documents\GitHub\worddoc\thesisImages\ttp.png | Trusted Third Party (TTP). Payment related logs are mostly to be generated by the TTP. |

Figure 1 shows the topology of the network and connections between entities.



Figure 1. Network Topology

Connection between serving access points is wireless, and they use IEEE 802.11s protocol [6]. The mesh backbone emulates a cloud from the mobile user’s perspective. It is a black box; which receives packets from mobile user and delivers them to the gateway in a multi-hop manner. Mesh backbone uses Hybrid Wireless Mesh Protocol (HWMP) [7], which is a hybrid routing protocol, which has routing tables.

Connection medium between mesh backbone and gateway (GW) is wireless. GWs and operators communicate through wired connection. The connection between an operator and TTP is also wired. These connections use 802.3(Ethernet protocol) [17].

1. Notations

The symbols and operators used in this paper are listed in Table 2.

TABLE II

System Entities

|  |  |
| --- | --- |
|  | XOR operation |
|  | Concatenation |
|  | Encryption of using the key |
|  | Decryption of using the key |
|  | Taking hash of times |
|  | Taking HMAC of using the key |
|  | th element of the hash chain (usage order) |
|  | Public key of TTP |
|  | Private key of TTP |
|  | th Access Point or its identity |
|  | th Operator or its identity |
|  | Public key of |
|  | Private key of |
|  | Serial Number |
|  | Nonce created by entity |
|  | Previous Alias |
|  | New Alias |
|  | Public key certificate of |
|  | Initialization Vector |
|  | Timestamp |
|  | Connection Request |
|  | Disconnection Request |
|  | Roaming Request |
|  | Change Alias Request |
|  | Mobility Request |
|  | Response (used in various protocol as positive acknowledgment) |
|  | Disconnection Acknowledgement |
|  | Roaming Acknowledgement |
|  | Mobility Response |
|  | Anonymized Subhash Chain |
|  | Length of Anonymized Subhash Chain |
|  | Change Alias Interval |

1. Connection Card Structure

*Connection Card* is the main deed that clients buy from the TTP and use to get Internet service. We use a prepaid system, in which connection cards include tokens for credit generation. Please note that the tokens in the connection card are not directly used to pay for the Internet service, but to generate credits to pay for the Internet service. Hash tokens are generated using hash chains as discussed below. Connection cards also have unique *Serial Numbers* (), which are to be used for alias computation.

Tokens are basically links in a hash chain. For each set of tokens, the TTP picks on a random and takes hashes of it many times. The number of hash operations is actually the number of token in a set. For example, if the client wants a hundred hash tokens, then the hash of is taken hundred times. More formally a hash chain with 100 tokens is constructed in the following way.

…

is the first token in the chain. The client uses this hash token to form a connection request for TTP. The generation of credits is explained in the following section.

Connection Cards are refillable with hash tokens, which are to be sold by the TTP. Operators compete with each other to provide high-quality service for broadband access in the WMN since the users are assumed to have free roaming.

Serial Number is a 128-bit value. With this setting, the system is able to support up to users. Hash tokens are to be generated using SHA-256 hash algorithm; hence they are 256-bit long.

Considering current technology, smart cards are suitable tools to be connection cards. A simple Connection Card with 4 KB memory could store a and approximately 1000 hash tokens.

1. Anonymized Subhash Chains

Clients change their aliases periodically to make their actions unlikable to their aliases. However, an adversary could trace a client’s actions by tracing the link between the hash tokens of the client.

To provide full untraceability in the system, clients form up anonymized subhash chains. The client and the TTP guesses the amount of the hash tokens that will be used in the next session. The Change Alias Interval () and Hash Token Renewal Interval () determine the Length of Anonymized Subhash Chains () as following:

The generation of the anonymized subhash chains is depicted in Figure 2.



Figure 2. Generation of Anonymized Subhash Chains with  
 and

Before any authentication or changing alias phase the client sends the first hash token of the remaining hash chain (In Figure 2 the first hash token is H0). TTP knows the value of the client. TTP and the client are able to form the anonymized subhash chain simultaneously. When the client sends the first hash token of the remaining hash chain to the TTP, TTP counts backwards from the received hash token times. Computes the corresponding anonymized hash token and takes the hash of the output times. These operations form up an anonymized subhash chain and the anonymized hash tokens are spent in reverse order.

In a case of a disconnection or connection drop before spending all the hash tokens in the anonymized subhash chain, client stores the index of the last used hash token index. For the next connection request the client sends the first hash token in the remaining hash chain to the TTP. For the new session, both the client and the TTP generate a new anonymized subhash chain.

In a mobility situation, clients transfer the next anonymized hash token to the new access point. The clients start to get service from the new access point when the transfer is finished.

An adversary could not relate two different anonymized subhash chains since using the hash output of XOR of a hash token and a random nonce value generates the hash chains. Every time a new anonymized subhash chain is generated a different nonce value is used. The hash operation on the seed of the anonymized subhash chain prevents any relation between different anonymized subhash chains. A demonstration of this scheme is as follows:

On the previous example it is infeasible to find the input by using or . Therefore, an adversary could not discover any hash token in the original hash chain by exploiting anonymized subhash chain because of the irreversibility property of hash algorithms. Moreover, and could not be related in feasible time. Therefore, usage of anonymized subhash chains provides unlinkability between different sessions.

1. Alias Computation

Aliases are temporary identifiers for clients. They change frequently using a secure protocol. Anonymity is achieved by changing aliases as previously stated way however it is durable to some extent.

The serial number (SN) of the CC, which is bought from an operator, will be used as a base for client’s aliases. An alias will be computed by performing the following operations:

1. Client will pick a random 128-bit unsigned number and call it his nonce .
2. Perform XOR operation with and his nonce; take the hash of the output.
3. Client will use this alias whenever his identity is required.

One may argue that this kind of alias computation would run a risk of producing same alias for several users. However making TTP to check the proposed alias to be a unique one solves this problem. This check is done in Change Alias protocol, which will be mentioned in Section 3.

1. Protocols

There exist ten protocols to make the system work. These protocols define packet transfers and routes. Cryptographic primitives and the way they are used are also explained in the protocol designs.

Some protocols show similarity e.g. *Initial Authorization* and *Reuse of a Connection Card*. The only difference between these two protocols is their hash token index. *Initial Authorization* uses the very first hash token while *Reuse of a Connection* Card using the other hash tokens on the hash chain. This kind of similar protocols will be explained simultaneously.

The designed protocols are formed by the usage of some cryptographic primitives such as public key cryptosystems and hash functions forms up the designed protocol. 2048-bit RSA [3] is employed for public key encryption-decryption and signature purposes. AES-128 [4] is utilized for symmetric key cryptography and SHA-256 [4, 5] is used as a hash algorithm in the system. HMAC [5, 6] algorithm is used for challenge-response protocols.

* 1. End-to-End Two-Way Protocols

The main protocol in the system is the End-to-End Two-way protocols, which are also the most common ones in the system. The generic depiction is shown in Figure 3.

The protocols classified as End-to-End Two-way are *Initial Authorization, Reuse of a Connection Card, Disconnection, Change Alias* protocols. These protocols transmit equally sized packets from client to TTP. TTP executes the same cryptographic operations on the packet and forwards the packet to the client. In these protocols client performs an encryption over a 384-bit packet using RSA-2048 and sends it to the TTP. TTP decrypts this cipher using RSA-2048 private key then signs 256-bit data using RSA-2048 private key. TTP sends this signed data to GW through the operator. GW encrypts the response with the symmetric key between itself and the target AP and sends it to the target AP through mesh backbone.

Macintosh HD:Users:canleloglu:Desktop:worddoc:thesisImages:protocolsInDetail:newPdf:end-to-end.pdf

Figure 3. End-to-End Two-Way Protocol Flow (OPT means optional)

Initial Authorization is the first protocol that a client uses in the system in order to get authorized. It is used only once by a particular user. Protocol starts with client forming up a Connection request. Considering the generic depiction Figure 3 in the case of Initial Authorization. Alias is calculated by taking the xor of Serial Number and a random nonce value as following . variable is . When TTP receives the Connection Request (CR) it decrypts it with it’s own private key and mark the client as connected in the database. In Initial Authorization protocol . When TTP receives the alias it first verifies the SN and hashes the Alias. TTP and client will calculate the alias that will be used in the system as following:.

Reuse of a Connection Card protocol is used when a user does not finish the tokens in a connection card and would like to use the remaining tokens at a later time. Initial Authorization and Reuse of a Connection Card protocols only differ in their hash token index. In Initial Authorization protocol the value is whereas, in Reuse of a Connection Card protocol holds where . In Initial Authorization and Reuse of a Connection Card protocols performing an XOR operation of SN with a random nonce forms a new Alias.

The initial time of the session for a user is stored when a user performs one of the two previously mentioned two protocols. Disconnection protocol yields the ending time of the session. In this way, the TTP learns the amount of time that the user got served. This information is used for settlement purposes. In Disconnection protocol . DR is formed as the same as a Connection Request the only difference is packet overhead, which determines the packet’s aim. There are 9 protocols that are used by the client; so 4-bit packet overhead is enough for this purpose. In Disconnection protocol client does not change it’s alias but uses the existing one. Therefore, in Disconnection protocol the client does not generate an anonymized subhash chain after sending the DR.

One of the privacy preserving features of the proposed system is that access points ask every user to change their aliases from time to time. When received such a command from the access point, clients compute aliases by calculating and send it to the TTP for signature and hash. The overall process is called Change Alias protocol. In this protocol the optional the packet request step is executed unlike the other protocols. Every active client forms up a Change Alias Request (CAR). In the case of Change Alias protocol . When TTP receives the CAR and it decrypts the content using it’s private key. Checks the last used hash token, if it is equal to the hash token that resides in the CAR then TTP signs the new Alias and the value. In this protocol TTP does not update client’s status in the database because Change Alias protocol keeps a connected client connected, thus an update is not necessary.

* 1. Access Point Authentication

Macintosh HD:Users:canleloglu:Desktop:worddoc:thesisImages:protocolsInDetail:newPdf:accessAuth.pdf

Figure 4. Access Point Authentication

*Access Point Authentication*, which is shown in Figure 4, takes place between a mobile client and an access point. It is a challenge-response type of protocol to authenticate the access point to the client.

*Access Point Authentication* starts with the serving access point by sending a request to the client. Client sends a 128-bit challenge to the access point. Access Point performs an HMAC [16] operation on this challenge using the last hash token as a key. Client performs the same operation and compares two results. If they match, the access point is verified as authenticated.

* 1. Packet Transfer

Macintosh HD:Users:canleloglu:Desktop:worddoc:thesisImages:protocolsInDetail:newPdf:packetTransfer.pdf

Figure 5. Packet Transfer

Packet Transfer protocol, shown in Figure 5, protocol is the simplest and the most commonly used protocol among others. It is the main service access protocol that uses tokens one by one. One token of the hash chain is sent from client to AP and the client starts to use the broadband access service. Usage is charged in time basis. Every five minutes client sends a new hash token to continue to get Internet service. When a user sends a hash token it means that she already has paid for the service and in case of disconnection the protocol is called after e.g. 2 minutes, user could not get a refund for the remaining 3 minutes.

The time measurement happens between access point and client. The access point does decrementing from 5 minutes. If client tries to get service after 5 minutes, access point sends a request to client to make her to send a new hash token.

* 1. Seamless Mobility and Roaming (Payment Related)

Macintosh HD:Users:canleloglu:Desktop:worddoc:thesisImages:protocolsInDetail:newPdf:seamlessRoaming.pdf

Figure 6. Seamless Mobility and Roaming

*Seamless Mobility* and *Roaming* protocols, shown in Figure 6, are run whenever the client changes the serving access point. The running protocol is called *Seamless Mobility* if the new access point belongs to the same operator as the previous access point. If the operators differ, then the protocol is called *Seamless Roaming.*

In these two protocols client sends a 384-bit request packet to the old access point. The old access point receives this packet and performs an encryption on it using RSA-2048, than signs this cipher text using RSA-2048 private key. The old access point sends this packet to client and the client relays it to the new access point. New access point decrypts the packet using RSA-2048 private key and verifies the signature using RSA-2048 public key.

Finally the new access point and the client run a *Challenge-Response Protocol* to authenticate the new access point.

If the running protocol is *Seamless Roaming*, then receiving break-off request from the client triggers the old access point to send a disconnection request to the TTP. This part of the protocol is not implemented in the unit test because it runs in background.

* 1. Update Packets

Macintosh HD:Users:canleloglu:Desktop:worddoc:thesisImages:protocolsInDetail:newPdf:updatePackets.pdf

Figure 7. Update Packets

*Update Packets* protocol, shown in Figure 7, is used in case of an unexpected behavior in network. If a client drops out of the network, operators and TTP needs to be informed that this client is not active anymore. In order to handle this unexpected behavior, the access points periodically update operators using Update Packets protocol.

In this protocol, client sends concatenation of 128-bit alias and 128-bit hash token to the operator. Operators update TTP in case of a drop. This protocol is a one way end-to-end protocol.

1. Payment to the Operators (Settlement)

In SSPayWMN, operators claim their money from the TTP by showing their service logs. A log proves a service that has been provided between a connection request and a disconnection request.

Operators store connection requests (CR) of the clients; CRs are formed in the Initial Authorization and Reuse of a Connection Card protocols. When a client makes a disconnection request, operator stores the disconnection request (DR) as well. After receiving the DR, operator forms its log as follows.

TS stand for timestamp in the logs. Timestamps are mandatory in the logs to make TTP’s job easier.

When TTP receives two consecutive logs from an operator:

1. TTP will sort the logs according to their TS value.
2. TTP first decrypts CR since it is encrypted with the public key of TTP. CR consists of Alias, Nonce and the first hash token to be used to get service.

Consider

TTP decrypts it using its private key, and gets SN by the XOR operation:

Note that SN’s first token used is Hf.

1. TTP decrypts the Signed Connection Response using its public key, and gets the alias and the hash token. TTP compares the values with the ones in connection request. If they match, then the log is marked as valid.
2. The abovementioned log is only a service starter; operator needs to show service-ending log to claim its money from the TTP.

Service ending log naturally has a larger TS value; therefore this log comes later in the sorted list of logs.

TTP takes the ending log and decrypts DR using its private key.

TTP gets Alias, Nonce and the hash token from the decrypted DR. TTP makes the XOR operation: and gets the SN. Note that SN used is the hash token came with the DR to end the service.

1. TTP takes the Signed Disconnection Response and decrypts it using its public key. TTP gets the alias and the hash token from it, and compares the values with the ones came with the DR. If the values match, TTP considers the log as a valid service-ending log.
2. After validating the logs, TTP performs the hash operation over service ending hash token until it reaches the service starter hash token. TTP counts these hash operations. This count is mapped to funds for the provided service.

However the misusage of the logs should be reckoned. Consider the situation of a client:

* Gets service from her home operator between H0 and H10
* Gets service from a foreign operator between H11 and H20
* Gets service from her home operator between H21 and H30

In this type of situation home operator has two CRs and DRs, whereas foreign operator has a CR and DR. Home operator has the following logs:

The home operator has served between H0 and H10and also has served between H21 and H30. Home operator would want to take the money for serving between H11 and H20. It could pretend that it has served the client between H11 and H20 by not sending Log2 and Log3. Since Log2 indicates that client is disconnected from the operator at H10 and Log3 suggests that the client started to get service from the operator at H21. Sending only Log1 and Log4 results TTP to think that the home operator has served the client between H0 and H30. This way operator would want money for serving 30 hash tokens.

Abovementioned situation suggests that there should be another operator, which has served between H11 and H20. Second operator would have two logs as follows.

Foreign operator proves that it has served between H11 and H20 by showing the signed RP and DA.

TTP would see that it has already paid home operator for service to that particular client between H11 and H20. This means that home operator has tricked TTP to pay more.

In the proposed system TTP is the one who has the authority, it pays operators their money. If the TTP finds an operator misbehaving it could give a penalty to the operator and do not pay for future services, or there could be several other kinds of penalties, since TTP has the proof it could bring the subject to the court as well.

1. Simulation Environment

The network topology is hierarchical and WMN supports connections with other IEEE 802.11 protocols [14, 15], clients communicate with TTP via APs, GWs and operators in sequence. Access points are connected to gateways with 6-54 Mbps Wi-Fi connection. Some important specifications about the *AP*s are shown in Table 3. *Update Interval* determines the time value between two update packets that access point send to TTP.

TABLE III

AP Specifications

|  |  |
| --- | --- |
| AP-Gateway Connection bit rate | 6-54Mbps – Wi-Fi |
| AP-Gateway Distance | 100m |
| Service Duration per token | 5minutes |
| Update Interval | 1 minutes |

The network consists of 32 gateways and 100 access points. In unit simulation there is only one mobile client whereas in real-life scenario simulations there are 100-300-500 mobile clients.

Public Key Operations and Their Timings

Public Key Cryptography timings for access points and gateways are mentioned in [11]. For operator servers and TTP servers, timings from [12] are used. For mobile clients, performance values from [23] are used, the performance values are for iPhone 4. For AES timings the values from [21] are used, which results a 0.00004 second of delay for AES on Linksys WRT54GS. The same value is used for gateways as well. Timings of hash algorithms are taken from [22].

Platform specifications are shown in Table 4, and RSA timings are shown in Table 5.

TABLE IV

Platform Specifications

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Gateway [11] | Linksys WRT54GS (AP) [11] | Server [12] | Client [23] |
| CPU Speed | 2.08 GHz | 200 MHz | Dual-core 64 bit 2.8 GHz | Not disclosed by Apple |
| CPU type | AMD Athlon XP 2800 | Broadcom MIPS32 | Intel Xeon | Arm Cortex-A8 |
| RAM | 512 MB | 32 MB | - | - |

TABLE V

RSA-2048 Timings

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Gateway [11] | Linksys WRT54GS [11] | Server [12] | Client [13] |
| RSA Signing | 47.3 ms | 1529.0 ms | 8.13 ms | 120 ms |
| RSA Verification | 1.3 ms | 37.9 ms | 0.32 ms | 3.3 ms |

1. Unit Test Results

Unit tests cover protocol behaviors under low pressure. In these tests there is only one user, and this user performs the same protocol every minute. These tests are done to ensure that modules of the system are fit for use.

As discussed earlier some protocols show similarity considering packet sizes, cryptographic operations and packet routes. Since there would be no difference between unit tests of protocols that are in the same group, there is one result chart for a particular group of protocols.

* 1. Results for End-to-End Two-Way Protocols

Unit tests for end-to-end two-way protocols consist of a user, running the same protocol every minute. End-to-end Two-way protocols consist of *Initial Authorization*, *Reuse-CC*, *Change Alias* and *Disconnection* protocols. Figure 8 presents the average delay of packet delivery over time. In this simulation the user sends the packet to a serving access point and the packet hops 2 times in the mesh backbone until it reaches the gateway. Gateway forwards the packet to operator and operator transmits the packet to TTP. TTP processes this packet and sends it back to the client through the same route.

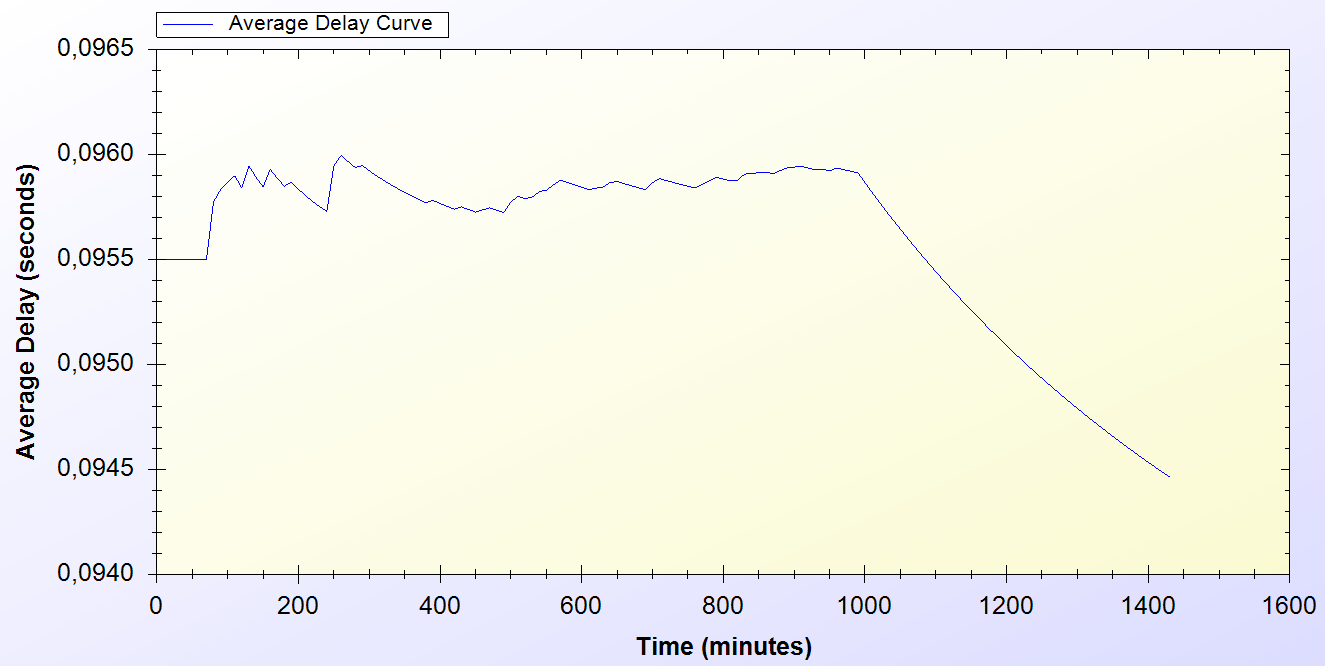


Figure 8. Unit Test Result for End-to-End Two-Way Protocols

As shown in Figure 8, there is a delay that shows variation around 0.09 second. This unstable behavior is caused by different initial packet delays. System needs some packets to set up paths between mesh nodes. The performance stabilizes in time. Average delay shows a peak by the end however the difference between highest and lowest values of the results is inconsiderable.

* 1. Results for Access Point Authentication Protocol

Access Point Authentication protocol, consists of a challenge-response protocol. It contains two HMAC operations.

Unit test for this protocol contains a user, trying to run access point authentication protocol with a serving access point every minute. The resulting chart, presented on Figure 9, shows the average delay of the protocol versus time.



Figure 9. Unit Test Result for Access Point Authentication Protocol

As shown in Figure 9, average delay of access point authentication converges to 0.05 second in the steady state. The initial delay values are higher than the later ones, because nodes need some time to establish and see who is around. At the time of initial deployment, wireless nodes send and receive beacons and perform operations using them.

* 1. Results for Seamless Mobility and Roaming Protocols

*Seamless Mobility* and *Seamless Roaming* protocols have the same behavior since client sends and receives same length of packets. Thus, they are grouped together for unit tests.

Unit test for Seamless Mobility and Seamless Roaming protocols consists of a client changes serving access point every minute. Client is located in between two access points and these access points are both eligible for service. Since these protocols must be seamless to the user it is important to get reasonable delays for these protocols.

Figure 10 presents the unit test result for Seamless Mobility and Roaming protocols.

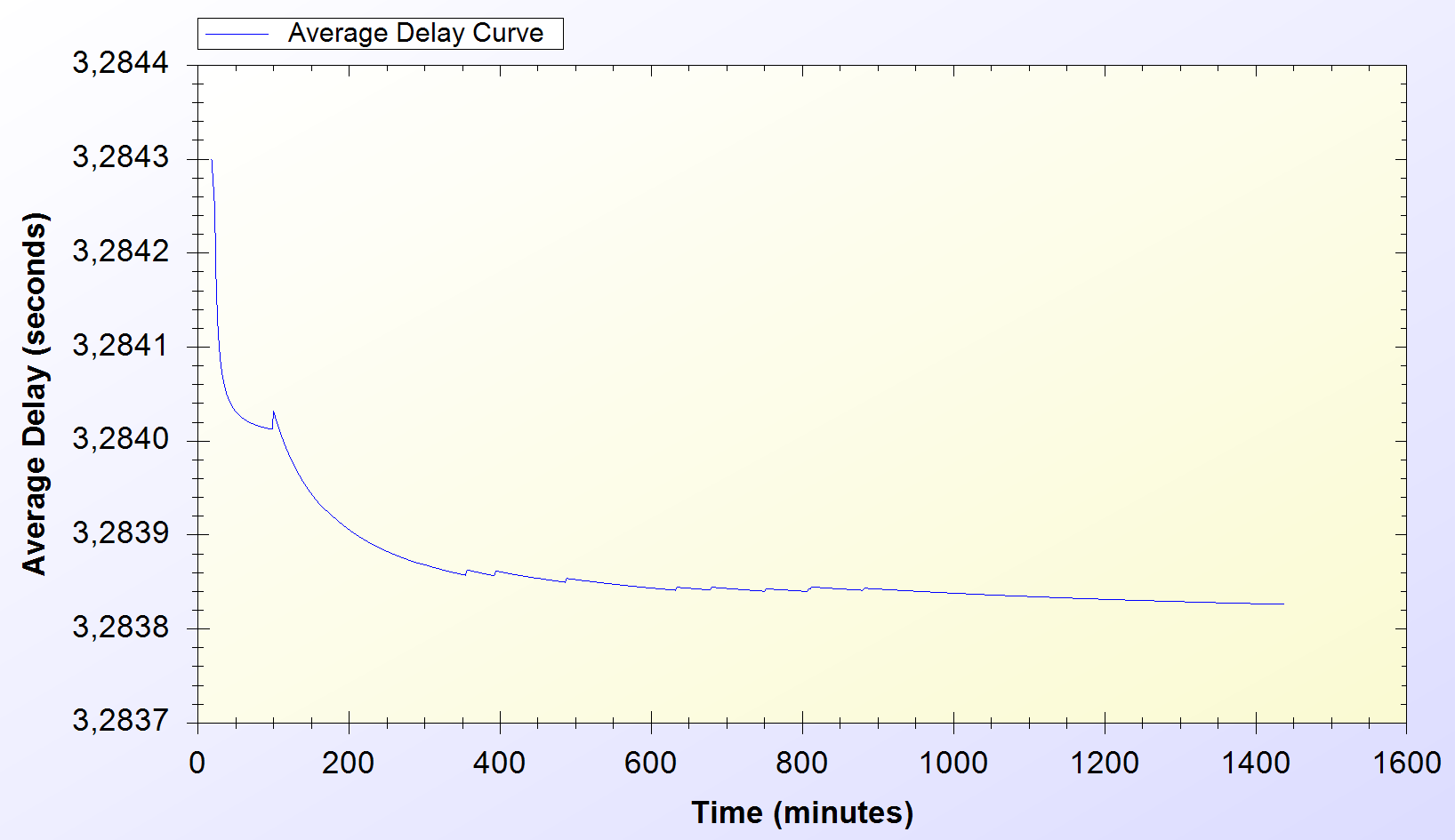


Figure 10. Unit Test Result for Seamless Mobility and Roaming Protocols

In unit test for these protocols, 3.28 seconds of average delay for anonymized subhash token transfer is observed. However the real network delay between service changes is approximately 0.2 second. Therefore seamless roaming and mobility is seamless to the clients. Similar to other protocols, there is a transitive period at the beginning of the simulations; however it reaches steady state in time.

* 1. Results for Packet Transfer Protocol

Packet transfer is the mostly used protocol in the system. It is crucial to have small amount of network delay for this protocol because of it’s often use. Packet transfer unit test scenario is that a client sends a 512-byte packet every minute.

Figure 11 shows the unit test result for Packet Transfer protocol.

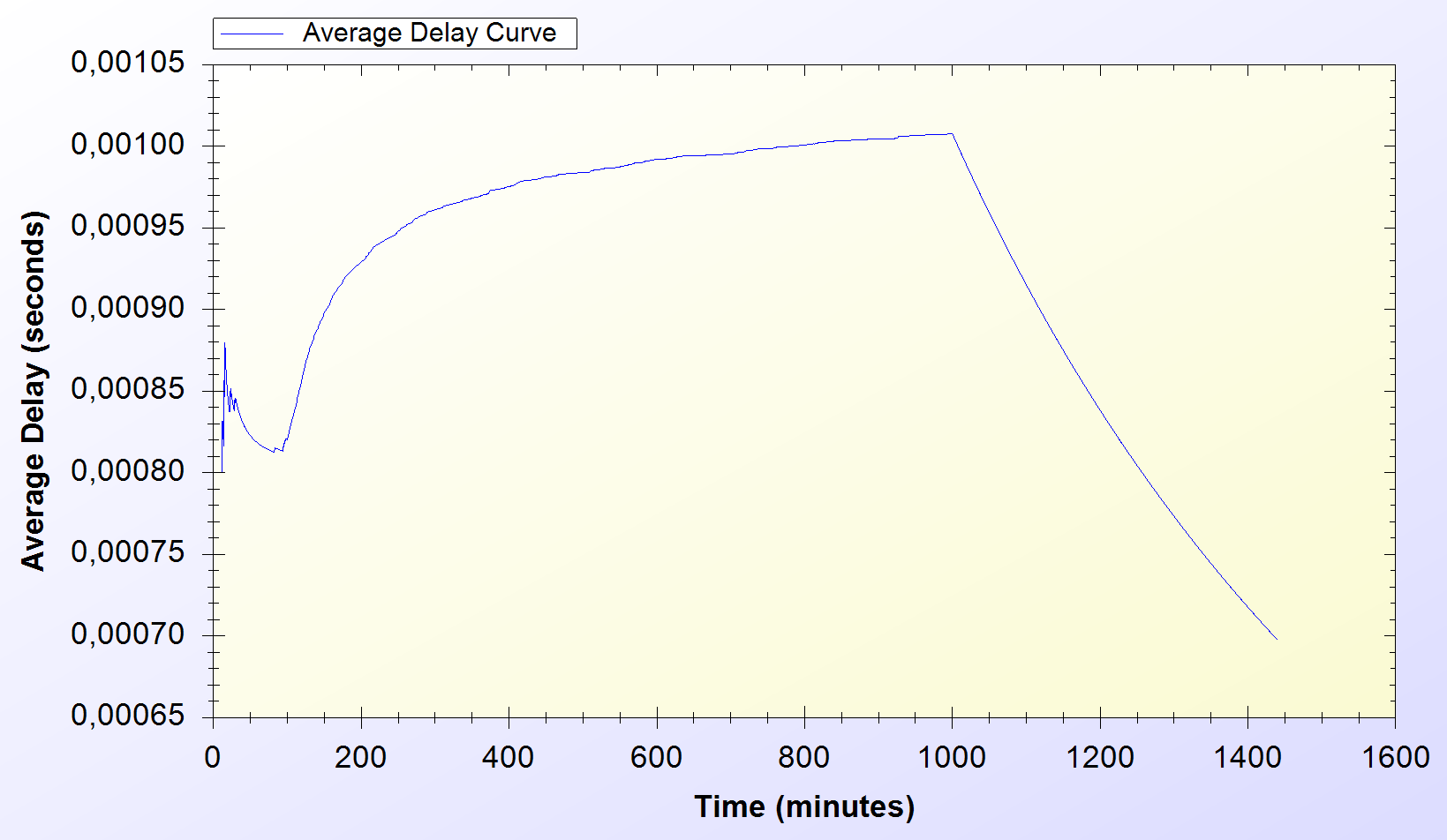


Figure 11. Unit Test Result for Packet Transfer Protocol

Unit test gave a higher average delay value at the early parts of the simulation but expectedly it reaches a balance through time. As seen on Figure 11, at steady state, packets are received in a very short amount of time, which is around 0.0008 second.

* 1. Results for Update Packets Protocol

Update Packets protocol takes place between AP and TTP. In this simulation access point updates the user info stored at operator. Figure 12 shows the average delay of Update Packets protocol over time.

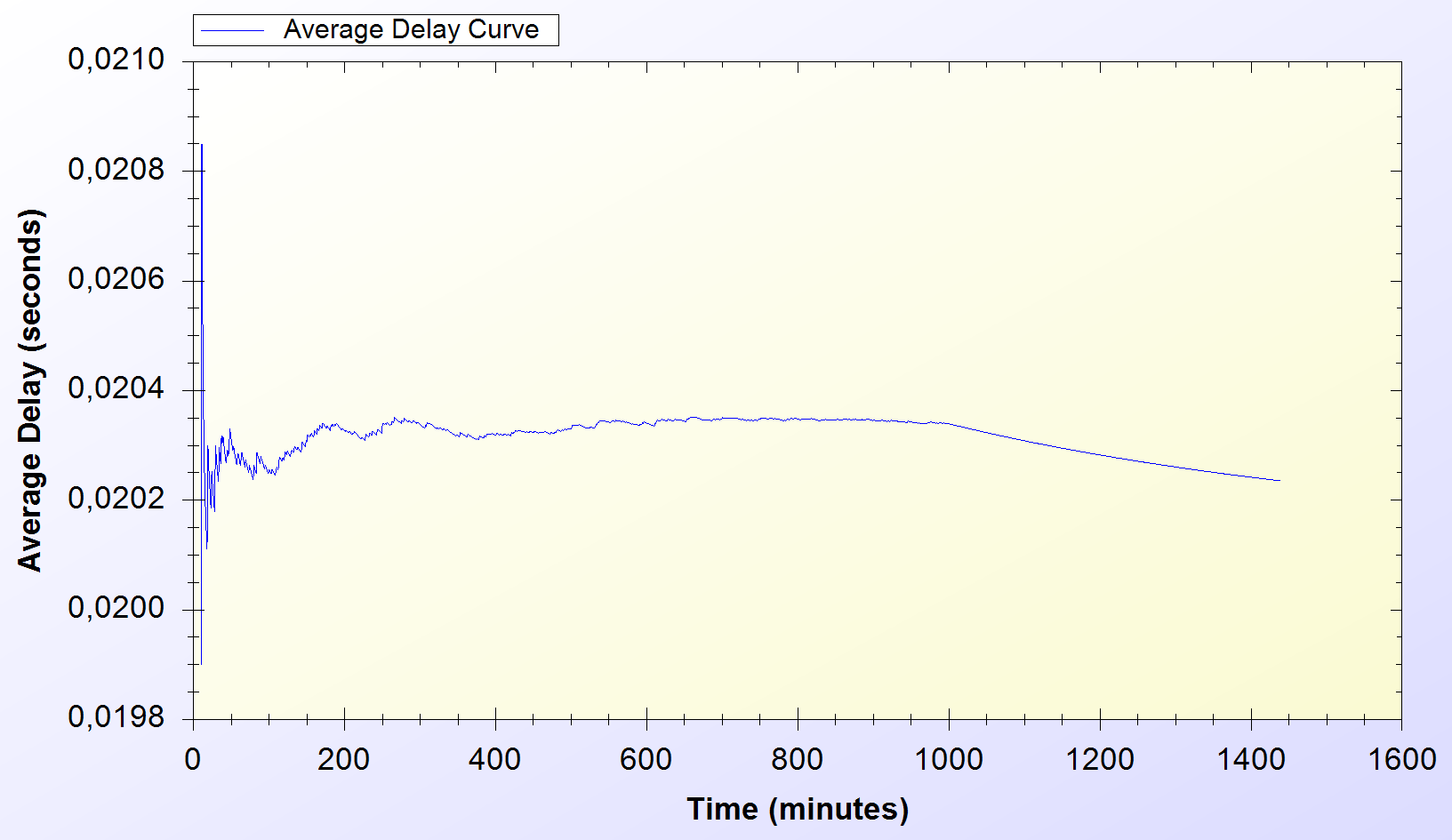


Figure 12. Unit Test Result for Update Packets Protocol

In the simulation scenario, APs update operator ten times in every second. Our simulation showed that there is a 0.02 second maximum network delay for updating operator for the client usage.

1. Discussions

In this section the properties of SSPayWMN are discussed.

*Seamless Roaming/Mobility*: Clients could continue getting service without an interruption in a case of handover.

*Anonymity*: For legal purposes users must give their identities to Trusted Third Party (TTP) for getting connection cards. Therefore, as far as *TTP* keeps clients’ identities secret, users can stay anonymous.

*Mutual authentication*: Challenge-Response Protocol ensures mutual authentication between AP and the client.

*No ultimate trust to operators*: Because of the one-way property of hash chains only the user could know the next element in the hash chain of tokens. Therefore without client giving the next element of the hash chain operator could not guess the element. The client could object to any type of over charge with cryptographic proofs.

*Preventing double spending*: All the connection card information is stored in the TTP’s database. TTP authorizes every token; it is not possible for client to use a token for a second time. Since TTP could not get the new token with a series of hash operations.

*Unlinkability*: SSPayWMN provides unlinkability by changing aliases periodically. Clients are traceable between the times they change their aliases nonetheless they could not be related to future actions after the alias change. The period of time to change the aliases is a choice of the system designer. In real-life scenario simulations the time period was 50 minutes.

1. Conclusions

In unit tests, standalone performances of the protocols under trivial usage scenarios are analyzed. The unit tests set an example for how the system will behave in empty hours. In this way, the first proof-of-concept implementation of the system is provided and it is demonstrated that the designed protocols reach steady state and reasonable performance in time.

The results are significant since the actual usage of the system is a combination of these protocols. Unit tests show that the proposed system is a considerable and an effective pre-payment system.

References

1. Akyildiz, I. F., Wang, X., and Wang, W. (2005) Wireless mesh networks: a survey, *Computer Networks and ISDN Systems*, 47(4): 445-487.
2. Network Simulator 3 Official Web Site: <http://www.nsnam.org/>
3. Rivest, R., Shamir, A., and Adleman, L. (1978) A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, 21(2): 120–126
4. FIPS PUB 197 (2001) Announcing the Advanced Encryption Standard (AES), <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
5. Trappe, W., and Washington, L. (2006) Introduction to cryptography with coding theory, *Person Education, Inc*
6. Joseph D. Camp and Edward W. Knightly The IEEE 802.11s Extended Service Set Mesh Networking Standard
7. Kai Yang, Jian-feng Ma, Zi-hui Miao (2009) Hybrid routing protocol for wireless mesh network, Computational Intelligence and Security – CIS ’09
8. Zaghloul, S., Bziuk, W. and Jukan, A. "A scalable Billing Architecture for Future Wireless Mesh Backhauls", IEEE ICC ’08.
9. Zhang, Y. and Fang, Y., "A secure authentication and billing architecture for wireless mesh networks", Wireless Networks, vol.13, no. 5, pp. 663-678, October 2007.
10. Lamport,L. (1981) Password authentication with insecure communication, *Proceedings of Commun. ACM,* vol. 24, no. 11,pp. 770-772.
11. Efstathiou, E., Frangoudis,P., and Polyzos,G. (2006) Stimulating Participation in Wireless Community Networks, *IEEE INFOCOM, 2006,* Barcelona*,* Spain.
12. Deng, L., and Kuzmanovic, A., (2009) A feeder-carrier-based internet user accountability service, *Northwestern University Technical Report,*<http://networks.cs.northwestern.edu/susinet/TR-09-12.pdf>
13. Yakovyna, V., Fedasyuk, D., Seniv M., Bilas O. (2007) The performance testing of RSA algorithm software realization, *CAD Systems in Microelectronics*, *CADSM ’07*, pp. 390-392, Polyana, UKRAINE
14. Intel Inc., Multi-Hop Mesh Networks—a new kind of Wi-Fi network.
15. J. Walker, Wi-Fi mesh networks, the path to mobile ad-hoc. Available from<http://www.wi-fitechnology.com/Wi-Fi_Reports_and_Papers/Mesh_Networks_References.html>.
16. American Bankers Association, *Keyed Hash Message Authentication Code*, ANSI X9.71, Washington, D.C., 2000.
17. Chaum, D. (1982) Untraceable electronic mail, return addresses, and digital pseudonyms, *Communications of the ACM, 4(2)*.
18. Vaughan-Nichols S.J., (2004) Achieving wireless broadband with WiMax, *IEEEComputer*, vol. 37, no.6, pp. 10-13
19. The ZigBee Alliance. Available from: <http://www.zigbee.org>.
20. Rao, Y.S.; Wing-Cheong Yeung; Kripalani, A.; , "Third-generation (3G) radio access standards," *Communication Technology Proceedings, 2000. WCC - ICCT 2000. International Conference on* , vol.2, no., pp.1017-1023 vol.2, 2000  
    doi: 10.1109/ICCT.2000.890849
21. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson Performance Comparison of the AES Submissions, Proc. Second AES Candidate Conference, NIST, March 1999, 15-34.
22. Crypto++ 5.6.0 Benchmarks: <http://www.cryptopp.com/benchmarks.html>

APPENDIX 1

We have considered the clients of SSPayWMN to have iPhone 4 on their hands. The system causes network delays as well as computational delays.

OpenSSL is run to understand that how much time an iPhone 4 needs to finish a certain cryptographic task. OpenSSL is run on a MacBook by connecting the iPhone 4 to it. The result of the command “opensll speed” is presented below.

The highlighted values of the result are used.

*OpenSSL run on iPhone 4:*

Doing md2 for 3s on 16 size blocks: 113903 md2's in 2.93s  
Doing md2 for 3s on 64 size blocks: 59789 md2's in 2.96s  
Doing md2 for 3s on 256 size blocks: 20740 md2's in 2.94s  
Doing md2 for 3s on 1024 size blocks: 5722 md2's in 2.95s  
Doing md2 for 3s on 8192 size blocks: 741 md2's in 2.95s  
Doing md4 for 3s on 16 size blocks: 962184 md4's in 2.99s  
Doing md4 for 3s on 64 size blocks: 808365 md4's in 2.94s  
Doing md4 for 3s on 256 size blocks: 553565 md4's in 2.94s  
Doing md4 for 3s on 1024 size blocks: 251644 md4's in 2.96s  
Doing md4 for 3s on 8192 size blocks: 41204 md4's in 2.94s  
Doing md5 for 3s on 16 size blocks: 512656 md5's in 2.88s  
Doing md5 for 3s on 64 size blocks: 446107 md5's in 2.70s  
Doing md5 for 3s on 256 size blocks: 323702 md5's in 2.65s  
Doing md5 for 3s on 1024 size blocks: 169849 md5's in 2.79s  
Doing md5 for 3s on 8192 size blocks: 30073 md5's in 2.68s  
Doing hmac(md5) for 3s on 16 size blocks: 933276 hmac(md5)'s in 2.62s  
Doing hmac(md5) for 3s on 64 size blocks: 741506 hmac(md5)'s in 2.72s  
Doing hmac(md5) for 3s on 256 size blocks: 490370 hmac(md5)'s in 2.62s  
Doing hmac(md5) for 3s on 1024 size blocks: 202907 hmac(md5)'s in 2.63s  
Doing hmac(md5) for 3s on 8192 size blocks: 28168 hmac(md5)'s in 2.32s  
Doing sha1 for 3s on 16 size blocks: 466460 sha1's in 2.83s  
Doing sha1 for 3s on 64 size blocks: 498797 sha1's in 2.92s  
Doing sha1 for 3s on 256 size blocks: 298535 sha1's in 2.90s  
Doing sha1 for 3s on 1024 size blocks: 114257 sha1's in 2.94s  
Doing sha1 for 3s on 8192 size blocks: 16773 sha1's in 2.94s  
Doing sha256 for 3s on 16 size blocks: 577481 sha256's in 2.93s  
Doing sha256 for 3s on 64 size blocks: 350266 sha256's in 2.96s  
Doing sha256 for 3s on 256 size blocks: 158648 sha256's in 2.96s  
Doing sha256 for 3s on 1024 size blocks: 49984 sha256's in 2.93s  
Doing sha256 for 3s on 8192 size blocks: 6722 sha256's in 2.88s  
Doing sha512 for 3s on 16 size blocks: 120505 sha512's in 2.93s  
Doing sha512 for 3s on 64 size blocks: 121406 sha512's in 2.91s  
Doing sha512 for 3s on 256 size blocks: 43968 sha512's in 2.92s  
Doing sha512 for 3s on 1024 size blocks: 15084 sha512's in 2.94s  
Doing sha512 for 3s on 8192 size blocks: 2119 sha512's in 2.96s  
Doing rmd160 for 3s on 16 size blocks: 500321 rmd160's in 2.88s  
Doing rmd160 for 3s on 64 size blocks: 518314 rmd160's in 2.93s  
Doing rmd160 for 3s on 256 size blocks: 294776 rmd160's in 2.93s  
Doing rmd160 for 3s on 1024 size blocks: 108497 rmd160's in 2.92s  
Doing rmd160 for 3s on 8192 size blocks: 14321 rmd160's in 2.69s  
Doing rc4 for 3s on 16 size blocks: 8838055 rc4's in 2.96s  
Doing rc4 for 3s on 64 size blocks: 2573509 rc4's in 2.91s  
Doing rc4 for 3s on 256 size blocks: 670210 rc4's in 2.93s  
Doing rc4 for 3s on 1024 size blocks: 168703 rc4's in 2.92s  
Doing rc4 for 3s on 8192 size blocks: 20854 rc4's in 2.71s  
Doing des cbc for 3s on 16 size blocks: 1656140 des cbc's in 2.75s  
Doing des cbc for 3s on 64 size blocks: 425114 des cbc's in 2.83s  
Doing des cbc for 3s on 256 size blocks: 108385 des cbc's in 2.87s  
Doing des cbc for 3s on 1024 size blocks: 28718 des cbc's in 2.76s  
Doing des cbc for 3s on 8192 size blocks: 3510 des cbc's in 2.81s  
Doing des ede3 for 3s on 16 size blocks: 646206 des ede3's in 2.83s  
Doing des ede3 for 3s on 64 size blocks: 164355 des ede3's in 2.87s  
Doing des ede3 for 3s on 256 size blocks: 41628 des ede3's in 2.83s  
Doing des ede3 for 3s on 1024 size blocks: 10274 des ede3's in 2.87s  
Doing des ede3 for 3s on 8192 size blocks: 1241 des ede3's in 2.80s  
Doing aes-128 cbc for 3s on 16 size blocks: 2403683 aes-128 cbc's in 2.89s  
Doing aes-128 cbc for 3s on 64 size blocks: 649635 aes-128 cbc's in 2.81s  
Doing aes-128 cbc for 3s on 256 size blocks: 174904 aes-128 cbc's in 2.89s  
Doing aes-128 cbc for 3s on 1024 size blocks: 45134 aes-128 cbc's in 2.94s  
Doing aes-128 cbc for 3s on 8192 size blocks: 5175 aes-128 cbc's in 2.68s  
Doing aes-192 cbc for 3s on 16 size blocks: 2168292 aes-192 cbc's in 2.91s  
Doing aes-192 cbc for 3s on 64 size blocks: 600481 aes-192 cbc's in 2.93s  
Doing aes-192 cbc for 3s on 256 size blocks: 155498 aes-192 cbc's in 2.90s  
Doing aes-192 cbc for 3s on 1024 size blocks: 39192 aes-192 cbc's in 2.91s  
Doing aes-192 cbc for 3s on 8192 size blocks: 4777 aes-192 cbc's in 2.86s  
Doing aes-256 cbc for 3s on 16 size blocks: 1983041 aes-256 cbc's in 2.94s  
Doing aes-256 cbc for 3s on 64 size blocks: 540276 aes-256 cbc's in 2.90s  
Doing aes-256 cbc for 3s on 256 size blocks: 137713 aes-256 cbc's in 2.89s  
Doing aes-256 cbc for 3s on 1024 size blocks: 35393 aes-256 cbc's in 2.93s  
Doing aes-256 cbc for 3s on 8192 size blocks: 4384 aes-256 cbc's in 2.93s  
Doing aes-128 ige for 3s on 16 size blocks: 2375637 aes-128 ige's in 2.92s  
Doing aes-128 ige for 3s on 64 size blocks: 698695 aes-128 ige's in 2.94s  
Doing aes-128 ige for 3s on 256 size blocks: 183967 aes-128 ige's in 2.91s  
Doing aes-128 ige for 3s on 1024 size blocks: 46891 aes-128 ige's in 2.86s  
Doing aes-128 ige for 3s on 8192 size blocks: 5870 aes-128 ige's in 2.93s  
Doing aes-192 ige for 3s on 16 size blocks: 2129667 aes-192 ige's in 2.94s  
Doing aes-192 ige for 3s on 64 size blocks: 615957 aes-192 ige's in 2.90s  
Doing aes-192 ige for 3s on 256 size blocks: 159366 aes-192 ige's in 2.89s  
Doing aes-192 ige for 3s on 1024 size blocks: 40431 aes-192 ige's in 2.90s  
Doing aes-192 ige for 3s on 8192 size blocks: 4566 aes-192 ige's in 2.62s  
Doing aes-256 ige for 3s on 16 size blocks: 1874993 aes-256 ige's in 2.85s  
Doing aes-256 ige for 3s on 64 size blocks: 546698 aes-256 ige's in 2.93s  
Doing aes-256 ige for 3s on 256 size blocks: 143584 aes-256 ige's in 2.87s  
Doing aes-256 ige for 3s on 1024 size blocks: 36373 aes-256 ige's in 2.82s  
Doing aes-256 ige for 3s on 8192 size blocks: 4479 aes-256 ige's in 2.88s  
Doing idea cbc for 3s on 16 size blocks: 1616991 idea cbc's in 2.84s  
Doing idea cbc for 3s on 64 size blocks: 419827 idea cbc's in 2.85s  
Doing idea cbc for 3s on 256 size blocks: 107747 idea cbc's in 2.91s  
Doing idea cbc for 3s on 1024 size blocks: 26694 idea cbc's in 2.90s  
Doing idea cbc for 3s on 8192 size blocks: 3366 idea cbc's in 2.91s  
Doing rc2 cbc for 3s on 16 size blocks: 1456945 rc2 cbc's in 2.86s  
Doing rc2 cbc for 3s on 64 size blocks: 387618 rc2 cbc's in 2.92s  
Doing rc2 cbc for 3s on 256 size blocks: 98467 rc2 cbc's in 2.89s  
Doing rc2 cbc for 3s on 1024 size blocks: 24847 rc2 cbc's in 2.92s  
Doing rc2 cbc for 3s on 8192 size blocks: 3091 rc2 cbc's in 2.95s  
Doing blowfish cbc for 3s on 16 size blocks: 3588480 blowfish cbc's in 2.92s  
Doing blowfish cbc for 3s on 64 size blocks: 969771 blowfish cbc's in 2.92s  
Doing blowfish cbc for 3s on 256 size blocks: 244745 blowfish cbc's in 2.89s  
Doing blowfish cbc for 3s on 1024 size blocks: 63589 blowfish cbc's in 2.97s  
Doing blowfish cbc for 3s on 8192 size blocks: 7246 blowfish cbc's in 2.65s  
Doing cast cbc for 3s on 16 size blocks: 2605484 cast cbc's in 2.93s  
Doing cast cbc for 3s on 64 size blocks: 689905 cast cbc's in 2.88s  
Doing cast cbc for 3s on 256 size blocks: 158869 cast cbc's in 2.80s  
Doing cast cbc for 3s on 1024 size blocks: 44980 cast cbc's in 2.90s  
Doing cast cbc for 3s on 8192 size blocks: 6318 cast cbc's in 2.92s  
Doing 512 bit private rsa's for 10s: 2657 512 bit private RSA's in 9.77s  
Doing 512 bit public rsa's for 10s: 29891 512 bit public RSA's in 9.75s  
Doing 1024 bit private rsa's for 10s: 507 1024 bit private RSA's in 9.81s  
Doing 1024 bit public rsa's for 10s: 9986 1024 bit public RSA's in 9.80s  
Doing 2048 bit private rsa's for 10s: 79 2048 bit private RSA's in 9.53s  
Doing 2048 bit public rsa's for 10s: 2934 2048 bit public RSA's in 9.85s  
Doing 4096 bit private rsa's for 10s: 13 4096 bit private RSA's in 10.50s  
Doing 4096 bit public rsa's for 10s: 818 4096 bit public RSA's in 9.70s  
Doing 512 bit sign dsa's for 10s: 3096 512 bit DSA signs in 9.80s  
Doing 512 bit verify dsa's for 10s: 2681 512 bit DSA verify in 9.78s  
Doing 1024 bit sign dsa's for 10s: 1004 1024 bit DSA signs in 9.58s  
Doing 1024 bit verify dsa's for 10s: 877 1024 bit DSA verify in 9.78s  
Doing 2048 bit sign dsa's for 10s: 297 2048 bit DSA signs in 9.68s  
Doing 2048 bit verify dsa's for 10s: 249 2048 bit DSA verify in 9.73s  
OpenSSL 0.9.8k 25 Mar 2009  
built on: date not available  
options:bn(64,32) md2(int) rc4(ptr,char) des(idx,cisc,16,long) aes(partial) idea(int) blowfish(ptr)   
compiler: arm-apple-darwin9-gcc -fPIC -fno-common -DOPENSSL\_PIC -DOPENSSL\_THREADS -D\_REENTRANT -DDSO\_DLFCN -DHAVE\_DLFCN\_H -D\_\_DARWIN\_UNIX03 -O3 -fomit-frame-pointer -fno-common  
available timing options: TIMEB USE\_TOD HZ=100 [sysconf value]  
timing function used: getrusage  
The 'numbers' are in 1000s of bytes per second processed.  
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes  
md2                622.00k     1292.74k     1805.93k     1986.21k     2057.72k  
mdc2                 0.00         0.00         0.00         0.00         0.00   
md4               5148.81k    17597.06k    48201.58k    87055.22k   114810.60k  
md5               2848.09k    10574.39k    31270.83k    62338.84k    91924.63k  
hmac(md5)         5699.40k    17447.20k    47914.02k    79002.57k    99462.18k  
sha1              2637.23k    10932.54k    26353.43k    39795.64k    46736.20k  
rmd160            2779.56k    11321.53k    25755.17k    38048.26k    43612.50k  
rc4              47773.27k    56599.51k    58557.60k    59161.60k    63039.10k  
des cbc           9635.72k     9613.89k     9667.79k    10654.79k    10232.71k  
des ede3          3653.46k     3665.06k     3765.64k     3665.71k     3630.81k  
idea cbc          9109.81k     9427.69k     9478.77k     9425.74k     9475.69k  
seed cbc             0.00         0.00         0.00         0.00         0.00   
rc2 cbc           8150.74k     8495.74k     8722.34k     8713.47k     8583.55k  
rc5-32/12 cbc        0.00         0.00         0.00         0.00         0.00   
blowfish cbc     19662.90k    21255.25k    21679.83k    21924.29k    22399.71k  
cast cbc         14227.90k    15331.22k    14525.17k    15882.59k    17725.02k  
aes-128 cbc      13307.59k    14795.96k    15493.23k    15720.14k    15818.51k  
aes-192 cbc      11921.88k    13116.31k    13726.72k    13791.27k    13682.93k  
aes-256 cbc      10792.06k    11923.33k    12198.80k    12369.43k    12257.25k  
camellia-128 cbc        0.00         0.00         0.00         0.00         0.00   
camellia-192 cbc        0.00         0.00         0.00         0.00         0.00   
camellia-256 cbc        0.00         0.00         0.00         0.00         0.00   
sha256            3153.48k     **7573.32k**    13720.91k    17468.81k    19120.36k  
sha512             658.05k     2670.10k     3854.73k     5253.75k     5864.48k  
aes-128 ige      13017.19k    15209.69k    16184.04k    16788.95k    16411.96k  
aes-192 ige      11590.02k    13593.53k    14116.85k    14276.33k    14276.59k  
aes-256 ige      10526.28k    11941.53k    12807.49k    13207.78k    12740.27k  
                  sign    verify    sign/s verify/s  
rsa  512 bits 0.003677s 0.000326s    272.0   3065.7  
rsa 1024 bits 0.019349s 0.000981s     51.7   1019.0  
rsa 2048 bits **0.120633s 0.003357s**      8.3    297.9  
rsa 4096 bits 0.807692s 0.011858s      1.2     84.3  
                  sign    verify    sign/s verify/s  
dsa  512 bits 0.003165s 0.003648s    315.9    274.1  
dsa 1024 bits 0.009542s 0.011152s    104.8     89.7  
dsa 2048 bits 0.032593s 0.039076s     30.7     25.6