Implementation and Performance Evaluation of EAP-TLS-KS

Jared Cordasco, Ulrike Meyer, and Susanne Wetzel
Stevens Institute of Technology
Hoboken, New Jersey 07030
Email: {jcordasc,umeyer,swetzel}@cs.stevens.edu

***Abstract*— In this paper we analyze the performance of authentication protocols for roaming in 802.11i-protected WLANs. In particular, we compare the recently introduced EAP-TLS-KS protocol to standard configurations in EAP-TLS. Roaming configurations for EAP-TLS are such that all traffic is forwarded to the home network leaving the foreign network no control over the authentication. Alternatively, the foreign network handles authentication on its own, and the home network relinquishes control. In contrast, EAP-TLS-KS involves both networks and gives each of them control over the authentication. In addition to performance evaluations, we discuss how to implement EAP-TLS-KS, what difficulties one may encounter, and how they can be solved.**

I. Introduction

As public area WLAN hotspot deployment increases, users must choose from a plethora of Wireless Internet Service Providers (WISPs). However, no single WISP can provide coverage in every desired hotspot location. To address this, WISPs have already started to establish roaming agreements that allow their customers to use other networks. For this to work, WISPs need methods for authenticating roaming users. The latest security standard, 802.11i [13], allows for the integration of such roaming authentication methods, which may be based on public key certificates or other credentials.

While public key based methods could allow a mobile device (MD) to authenticate to a foreign network (FN) without its home network (HN) being involved, in practice this is seldom done as the user's HN wishes to control each authentication. In addition, authentication between MD and FN (without HN) requires that HN issue a separate roaming certificate for each FN its users will roam to. This requires MD to perform offline certificate chain validation, which is quite difficult. For these reasons, current roaming practices give HN full control over the authentication while FN simply proxies authentication traffic and then provides network access afterwards. However, a secure channel is currently required for all roaming authentication methods that simply proxy authentication traffic through FN to HN.

EAP-TLS-KS was introduced in [18] to remedy the problems associated with authenticating to either FN or HN. EAP-TLS-KS is based on EAP-TLS and uses key splitting, distributed decryption, and distributed signatures to give both HN and FN control over the authentication. In contrast to proposed public key based methods that do not involve HN in the authentication, a HN using EAP-TLS-KS has a single roaming certificate. For each FN, HN creates a unique key split of the private key and issues the FN a part of this split. The use of a single roaming certificate prevents MD from having to deal with offline certificate chain validation operations as the single roaming certificate can be preinstalled by HN when the device is first configured. In addition, the use of unique splits for each FN allows FN's identity to be proven to MD. Finally, the use of key splitting eliminates the need for a secure channel between FN and HN for key transfer.

In addition to these advantages, EAP-TLS-KS was shown to provide theoretical performance improvements over EAP-TLS due to a reduction in the number of round-trip message exchanges required between FN and HN.

The main contribution of this paper is to show that EAP-TLS-KS does in fact outperform EAP-TLS in practice. The performance of both protocols is compared by implementing and testing them in local, national, and international roaming scenarios. The paper includes a detailed description of the changes necessary to turn an existing EAP-TLS implementation into an EAP-TLS-KS implementation, as well as the presentation and analysis of the experimental results.

II. Related Work

EAP was introduced for use with the PPP protocol in [5] by Blunk et al. In [1], Aboba et al. introduced the TLS extension for EAP. Rigney et al. created the current RADIUS standard in [21]. Then, Rigney et al. [20] developed EAP as a RADIUS extension. Stanley et al. define requirements for EAP methods used in IEEE 802.11 wireless LAN in [24].

Various other PKI-based protocols have been proposed for inter-provider roaming which authenticate both MD and the network it is attaching to. In [4], Bayarou et al. suggest an authentication framework for wireless networks. Gu et al. propose an authentication method specifically for WLAN in [11]. Both of these protocols require an offline MD to discover, verify, and validate certificate chains. In [4], this problem is addressed by deligating the discovery and validation to a trusted server. This requires communication with the trusted server in addition to the protocol itself. In contrast to the above proposals, EAP-TLS-KS does not require MD to discover, verify, or validate certificate chains, nor does it delegate this to a trusted server. MD simply compares the certificate it receives to the roaming certificate that was preinstalled.

Salgarelli et al. [22] propose an efficient authentication protocol, EAP-W-SKE, in which efficiency is gained by requiring only one round-trip message exchange between HN and FN. However, this also requires that a session key be transferred over a secure channel established between HN and FN. As opposed to EAP-W-SKE, EAP-TLS-KS does not require a secure channel between FN and HN for key transfer.

Matsunaga et al. suggest an architecture to be combined with web-based authentication mechanisms in [16]. Their approach combines several layers of security. They present authentication time results for each layer along with that of the entire mechanism for both local and roaming authentications. It is unclear, however, how the timings for the different layers affect each other.

Agarwal et al. [2], [3] provide performance comparisons between EAP-MD5 and EAP-TLS with and without IPsec. It is unclear at which points in the authentication they start and stop measurement. In [19], Prasithsangaree et al. use estimates of network latency and battery consumption along with the total number of messages sent to compare the authentication times and the energy usage of EAP-AKA, EAP-TLS-RSA, EAP-TLS-ECC, and EAP-TTLS-RSA. None of these papers studies the effects of different roaming scenarios on the performance of the authentication procedures. This paper provides such a study for EAP-TLS as well as EAP-TLS-KS and is the first comparative evaluation involving EAP-TLS-KS.

III. EAP-TLS AND EAP-TLS-KS

A. Overview of EAP-TLS for roaming users

EAP-TLS is an EAP-Method defined in RFC 2716 [1]. It supports either mutual or server-side authentication between a client and a server.

One application for EAP-TLS is for mutual authentication between a roaming MD and the authentication server (AS) of a WLAN network in the IEEE 802.11i security framework. In this case, MD is subscribed to one particular WLAN network provider, HN, who stores all user related subscription data. HN and MD are both in possession of a public key certificate signed by a Certification Authority (CA) both trust. Upon roaming to a FN, that has a roaming agreement with MD's HN, FN and MD can authenticate each other using EAP-TLS based on the certificates of MD and HN.

EAP-TLS supports three different types of certificates (certificates for RSA encryption, RSA signature verification, or DSS signature verification). If a certificate of the first type is used, the key exchange method is based on an RSA-encrypted random number. If a certificate of the second or third type is used, the key exchange is based on a Diffie-Hellman (DH) key exchange.

Figure 1 gives an overview of the EAP-TLS protocol between MD, FN, and HN. In this case, MD and HN each have a certificate including a public key for RSA signature verification and use the DH key exchange. This case is referred to as the DHE-RSA case. FN proxies all EAP messages between MD and HN until the EAP authentication terminates. In case of successful authentication, HN transfers the master session key exchanged during the authentication to FN. From this key, MD and FN can derive session keys to secure their subsequent communication.

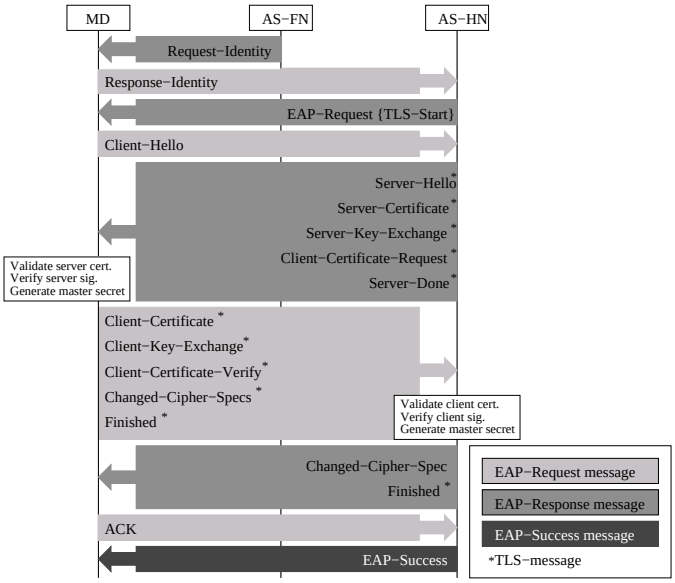

Fig. 1. Overview of EAP-TLS for roaming mobile devices.

FN sends an `EAP-Request` message to MD requesting MD's EAP-identity. MD answers with an `EAP-Response` message including its identity. HN answers with the `TLS-Start` message in an `EAP-Request` message and the TLS-Handshake begins: MD sends the `TLS-Client-Hello` message in an `EAP-Response` message to HN. `Client-Hello` includes a random number Client.RAND that guarantees the freshness of the resulting keys to MD. HN answers with an `EAP-Request` message including the TLS messages `Server-Hello`, `Server-Certificate`, `Server-Key-Exchange`, `Client-Certificate-Request`, and `Server-Done`. The `Server-Hello` message includes a random number Server.RAND that guarantees key freshness to FN. The `Server-Certificate` includes HN's RSA signature verification key. The `Server-Key-Exchange` includes HN's public DH parameter for this protocol instance. Moreover, a hash value of HN's public DH parameter concatenated with Client.RAND and Server.RAND is signed with HN's RSA signature key and included in the `Server-Key-Exchange`.

Upon receipt of these messages, MD computes a DH key from HN's public DH parameter and its own private DH parameter. MD then computes a master secret using Client.RAND, Server.RAND, and the DH key as input to a Pseudo-Random Number Generator (PRNG).

MD then sends an `EAP-Response` including the TLS messages `Client-Certificate`, ..., `Finish` to HN as illustrated in Figure 1. The `Client-Certificate` includes MD's public RSA signature verification key. The `Client-Key-Exchange` includes MD's public

DH parameter for this protocol instance. In order to prove its identity to HN, MD's response includes the `Client-Certificate-Verify` message. This message contains a hash value of all messages sent and received so far starting from `Client-Hello` up to and including the `Client-Key-Exchange`. The message is signed using MD's RSA signature key. The same `EAP-Response` message also includes `Change-Cipher-Spec` and `Finished`. With the former, MD indicates that it will now use the new ciphers and keys. The latter, which is protected with the new cipher-suite and keys, confirms that MD uses the same cipher-suite and keys as the server.

HN generates the DH key from its own private DH parameter and MD's public DH parameter. It then generates the master secret using the same method as MD. HN answers with the `Change-Cipher-Spec` and `Finished` messages. By verifying the correct encryption of the `Finished` message,[1] MD obtains a proof of HN's identity since only HN can generate the correct master secret. MD indicates successful receipt and verification by replying with an empty `EAP-Response` message. The EAP-TLS protocol ends with an `EAP-Success` message sent from HN to MD.

B. Overview of EAP-TLS-KS

In the standard usage of EAP-TLS for authentication between roaming MDs and foreign networks all authentication traffic is proxied by the AS in the foreign network to the AS in the home network. MD and HN each have a public key certificate which is the basis for the mutual authentication. As a result of the authentication, HN and MD share a secret master key. This master key is forwarded to FN over a secure channel between the two ASs. In contrast, EAP-TLS-KS has the advantage that the master secret is generated by FN and that FN is actively involved in the authentication of a roaming MD. As a consequence, no secure channel is required between HN and FN and the number of message exchanges between HN and FN is reduced.

In EAP-TLS-KS, each HN is issued a roaming certificate. The public key corresponding to this certficate is split between HN and each of its roaming partners. Each MD stores the roaming certificate of its HN.

The public roaming key is a pair (e, n) of an RSA modulus $n = pq$ with p, q prime and RSA signature verification key e with $\gcd(e, \varphi(n)) = 1$. The secret roaming key d is the inverse of e modulo $\varphi(n)$. The key splitting works as follows: let ℓ be the number of FNs to receive a share of the secret roaming key from HN. Then, HN splits the roaming key d into d_{HN_i}, d_{FN_i} for $i = 1, \ldots, \ell$ in the following way: HN randomly chooses $\omega_1, \ldots, \omega_\ell \in \mathbb{Z}_{\frac{\varphi(n)}{2}}$ such that $\omega_i \neq \omega_j$ for $i \neq j$. Then,

$$d_{HN_i} = d + 2\omega_i \mod \varphi(n), \quad i = 1, \ldots, \ell$$
$$d_{FN_i} = d + \omega_i \mod \varphi(n), \quad i = 1, \ldots, \ell.$$

[1]Like the `Client-Certificate-Verify` message, the `Finished` message is based on a hash of all messages in the protocol so far. The hash in the `Finished` message however is encrypted (not signed) with the new ciphers and keys.

Consequently, $d = -d_{HN_i} + 2d_{FN_i} \mod \varphi(n)$, for all $i = 1, \ldots, \ell$. HN distributes d_{FN_i} to FN_i and keeps copies of all ω_i as well as d.

HN and any single FN_i can now jointly sign a hash value $h(m)$ of a message m: HN generates a partial signature on $h(m)$ by computing $D_{d_{HN_i}}(h(m)) = h(m)^{-d_{HN_i}}$ and sends the result to FN_i. FN_i computes $s = h(m)^{-d_{HN_i}} h(m)^{2d_{FN_i}} = h(m)^{-d_{HN_i}+2d_{FN_i}} = h(m)^d$. Upon receipt of s and m, MD can check the signature by verifying that $h(m) = s^e \mod n$. This distributed RSA signature is used in the EAP-TLS-KS protocol as shown in Figure 2.

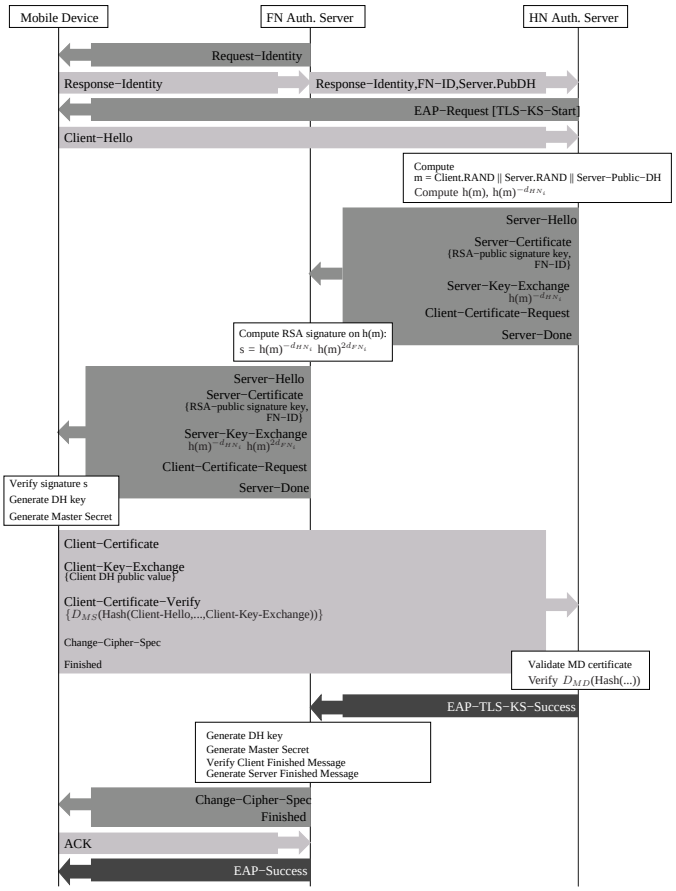

Fig. 2. Overview of EAP-TLS-KS with DHE-RSA.

The EAP-TLS-KS protocol starts as EAP-TLS does, with the `EAP-Request-Identity` message from FN. MD responds with an `EAP-Response-Identity` to which FN attaches its own identity, FN-ID, and its Server-DH value. This message is then forwarded to HN. HN replies with the `EAP-TLS-KS-Start` message.

FN then forwards the `Client-Hello` message to HN. HN computes the value $h(m)$, which is the hash of the concatenation of Client.RAND, Server.RAND, and its public DH parameter. Finally, HN computes $h(m)^{-d_{HN_i}}$ and includes this signature in the `Server-Key-Exchange` message. HN

sends the messages `Server-Hello`, ..., `Server-Done` to FN.

Upon receipt of these messages, FN uses its share d_{FN_i} of the secret roaming key to compute $s = h(m)^{-d_{HN_i}} h(m)^{2d_{FN_i}}$ and replaces $h(m)^{-d_{HN_i}}$ with s in the `Server-Key-Exchange` message. FN then fowards to MD the modified `Server-Key-Exchange` message together with the other (unchanged) EAP-TLS messages received from HN. FN computes the DH key as well as the secret master key from the private Server-DH value, the public DH value of MD, Client.RAND, and Server.RAND.

Upon receipt of the `Server-Key-Exchange` message, MD verifies the signature s with the public roaming key of HN and generates the DH key as well as the secret master key as in a regular EAP-TLS implemention. MD then sends the messages `Client-Certificate`, ..., `Finished` to FN. FN forwards the messages `Client-Certificate`, ..., `Client-Certificate-Verify` to HN. HN validates MD's certificate and—in the case of success—sends the `EAP-TLS-KS-Success` message to FN. Upon receipt of the `EAP-TLS-KS-Success` message, FN computes the DH key as well as the secret master key from the private Server-DH value, the public DH value of MD, Client.RAND, and Server.RAND.

The protocol concludes with the exchange of the messages `Changed-Cipher-Spec`, `Finished`, `Empty`, and `EAP-Success` between FN and MD without the need to involve HN. Because of the correctness of the encryption of the `Finished` message, MD is assured of FN's identity.

IV. Implementing EAP-TLS-KS

A. Testbed

The implementation and testing of EAP-TLS-KS comprises five components. The first three components: the MD, the AS for FN, and the AS for HN have already been introduced in Section III. The fourth component is the access point (AP) which forwards traffic between MD and FN's AS. The final component is a network monitor which observes the authentications and measures the authentication times. The network monitor, however, is not involved in the authentication process.

In order to obtain more comprehensive and realistic results, tests are run against three different HN ASs. One HN AS was located on campus about one quarter mile away from the FN AS. The ASs are connected over the campus' fiber optic backbone. This HN AS models a local roaming scenario. Another HN AS was located in Madison, Wisconsin, approximately 800 miles away from the FN AS on campus in New Jersey. This HN AS was connected through a residential DSL connection. Authentication with this HN AS models a national roaming scenario. Finally, the third HN AS was located in Germany, connected to a university network, thus providing an international roaming scenario.

In total, the testbed consisted of five seperate machines as seen in Figure 3. All but one of the machines were installed

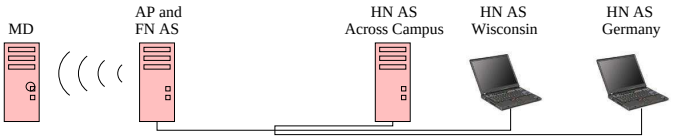

Fig. 3. Overview of the testbed setup.

with the Slackware 10.1 [23] distribution of Linux running kernel version 2.6.10.

The first of the three machines used was a desktop containing an AMD Athlon XP 1500+ processor, 512MB RAM, 100Mbps Ethernet, and a D-Link DWL-G510 PCI wireless card (hardware revision B1). This machine functioned as the access point, the foreign network authentication server, and the monitor of the network traffic on both the wired and wireless links. The necessary software installed for these functions included: hostapd v0.4.2 [12], madwifi drivers snapshot r1417-20060128 [15], FreeRADIUS* v1.1.0 [9] and ethereal* v0.10.9 [8]. [2]

The second machine was a desktop containing an AMD Athlon 1.2GHz, 256MB RAM, and a DemarcTech Reliawave 802.11b 100mW Prism 2.5 PC card (primary firmware version 1.0.7, secondary firmware version 1.7.4). This machine functioned as the wireless client. For this purpose, it also had installed the hostap drivers v0.4.7 and wpa_supplicant* v0.3.9 [12]. This machine and the first machine were in close proximity to each other (less than 5 feet).

The third machine was a virtual machine. The host machine contained an Intel Xeon 2.8GHz processor, 2GB of RAM, and 1000Mbps Ethernet. It ran the Gentoo [10] distribution of Linux and kernel version 2.6.10. The virtual machine was one of three setups on the machine and was limited to 300MB RAM usage. It also ran Gentoo with Linux kernel version 2.6.8. This machine served as the local HN AS. The fourth machine was an IBM T40 laptop with an Intel Pentium M 1.7GHz processor, 1GB RAM, and 1000Mbps Ethernet. It served as the national HN AS. The fifth machine was an IBM T43 laptop with an Intel Pentium M 2.0GHz processor, 512MB RAM, and 1000Mbps Ethernet. This laptop was the international HN AS. All three HN AS machines ran FreeRADIUS* v1.1.0.

B. Implementation

The approach to implementing EAP-TLS-KS authentication centers around modifying an existing RADIUS server. An EAP-Type value of 52 was introduced for EAP-TLS-KS to allow easy differentiation of EAP-TLS and EAP-TLS-KS traffic. This required a few simple modifications to the wpa_supplicant and ethereal software packages so that they would treat an EAP-Type of 52 the same as the standard EAP-Type for EAP-TLS (EAP-Type 13). It was not necessary to modify the hostapd software package as APs are EAP agnostic.

EAP-TLS-KS support was added to FreeRADIUS [9], a popular open source RADIUS server implementation. FreeRA-

[2]The asterisk following the name of a software package indicates that it was modified as described in Section IV-B.

DIUS has a modular design that allows for different authentication methods to be configured within a single server instance. Two such modules are the `realm` module which determines RADIUS proxying and the `eap` module which handles EAP authentication. The `eap` module shares several design patterns with the server itself, including the focus on modularity. Within the `eap` module, there are submodules for various types of EAP authentication such as EAP-MD5, EAP-PEAP, EAP-SIM, EAP-TLS, and EAP-TTLS. In order to add the implementation for EAP-TLS-KS, it was only necessary to make minor modifications to the `eap` module itself and then add a submodule to handle the specifics of EAP-TLS-KS for both HN and FN as shown in Figure 4.

As Figure 4 illustrates, the `eap` module simply had to be adjusted to provide pre-proxy and post-proxy callbacks. Then, the `eap` module could be added to the pre-proxy and post-proxy sections of the server configuration so that it would be called for packets being proxied to other servers. If the packet being proxied is not EAP-TLS-KS, then the callback simply returns without taking any action. However, if the packet is an EAP-TLS-KS packet, it is passed on to the EAP-TLS-KS submodule for processing, which results in either the packet being modified, or remaining the same. Through these calls into the EAP-TLS-KS submodule the changes for the FN side of the protocol were implemented.

The changes for HN were integrated into the authentication processing callback of the EAP-TLS-KS submodule. The base code for this part of the submodule was borrowed from the EAP-TLS submodule. Any functions that were not modified for use in EAP-TLS-KS are shared between the submodules. The description of the modifications necessary for both HN and FN proceeds in the order of execution of the protocol as illustrated in Figure 2.

EAP-Identity

The first of the changes is to append two attributes to the RADIUS packet containing the `EAP-Identity` message. These two attributes, the FN-ID and Server-Public-DH are added in the pre-proxy processing. The packet is then forwarded to HN which extracts the Server-Public-DH parameters and sets the SSL context's DH parameters from these. The HN also uses the FN-ID to select the key split to use.

Server-Key-Exchange

As the protocol proceeds, the `Client-Hello` message is caught by FN's pre-proxy processing in order to extract the value of Client.RAND. Similarly, the `Server-Hello` message is caught in the post-proxy processing to extract the value of Server.RAND. All of the extracted values are stored for use by FN later when completing the signature in the `Server-Key-Exchange` message. After the values are extracted, the messages are proxied to either HN or MD as necessary.

Since the EAP-TLS submodule uses OpenSSL to perform TLS negotiation, the EAP-TLS-KS submodule does the same whereever possible. A benefit of this is that OpenSSL builds a hash of all messages exchanged so far as per RFC 2246 [7]. Since this is used later in the `Finished` messages, it is preferable for the OpenSSL hash to be correct. For this reason, OpenSSL generates the `Server-Key-Exchange` message with the full RSA signature key at the HN as it normally would for EAP-TLS. The message is caught before being sent to FN and the partial signature is generated and substituted for the full signature.[3]

FN catches the `Server-Key-Exchange` message sent by HN containing the partial signature $h(m)^{-d_{HN_i}}$. It then calculates the partial signature $h(m)^{2d_{FN_i}}$ and multiplies this value with the value extracted from the message to complete the signature. FN replaces the partial signature in the message with the complete signature and forwards the message on to MD.

Partial signature generation

Generating the partial signatures is a challenging task. Since the OpenSSL library is used for TLS negotiation, all RSA signatures using private keys are performed with blinding enabled. This presents a problem when using the OpenSSL signature generation functions for partial signatures.

OpenSSL uses the following method for RSA blinding: First a random value $A \in \mathbb{Z}_n$ is chosen. Then, the multiplicative inverse A^{-1} as well as A^e are calculated. A^e and the message m are multiplied, the product is raised to the power d and finally multiplied by A^{-1}. This yields

$$(A^e \cdot m)^d \cdot A^{-1} = (A^{e \cdot d} \cdot m^d) \cdot A^{-1} = A \cdot m^d \cdot A^{-1} = m^d$$

which is the signature of the message m with key d. As shown in [6], this blinding prevents an attacker from performing a timing attack as the attacker does not know the value $A^e \cdot m$ that is raised to the power d. Unfortunately, this method does not work with key splits unless all parties involved agree on the same value for A. In the future, a suitable blinding method should be developed which protects against timing attacks on key splits.

In order to compare the existing EAP-TLS implementation with the EAP-TLS-KS implementation (when generating partial signatures) OpenSSL's blinding is disabled and the same calculations as in the blinding process are performed independent of the OpenSSL signature call. The code to compensate for blinding generates A and computes A^e as well as A^{-1} exactly as the blinding in OpenSSL would. It then performs a multiplication by A^e and a multiplication by A^{-1} calculating $A^e \cdot m \cdot A^{-1}$. The only computation remaining is the exponentiation by d. This occurs in the call for OpenSSL to sign m under d with blinding disabled. In this way, each of the partial signatures performed with a key split is comparable to a normal signature using OpenSSL.

Finished messages

In response to the `Server-Key-Exchange` message, MD sends the `Client-Key-Exchange` message which FN catches in order to extract MD's public DH parameters. FN also catches the `Change-Cipher-Spec` and `Finished`

[3]Both signatures would have to be generated regardless of how the hash is being constructed.

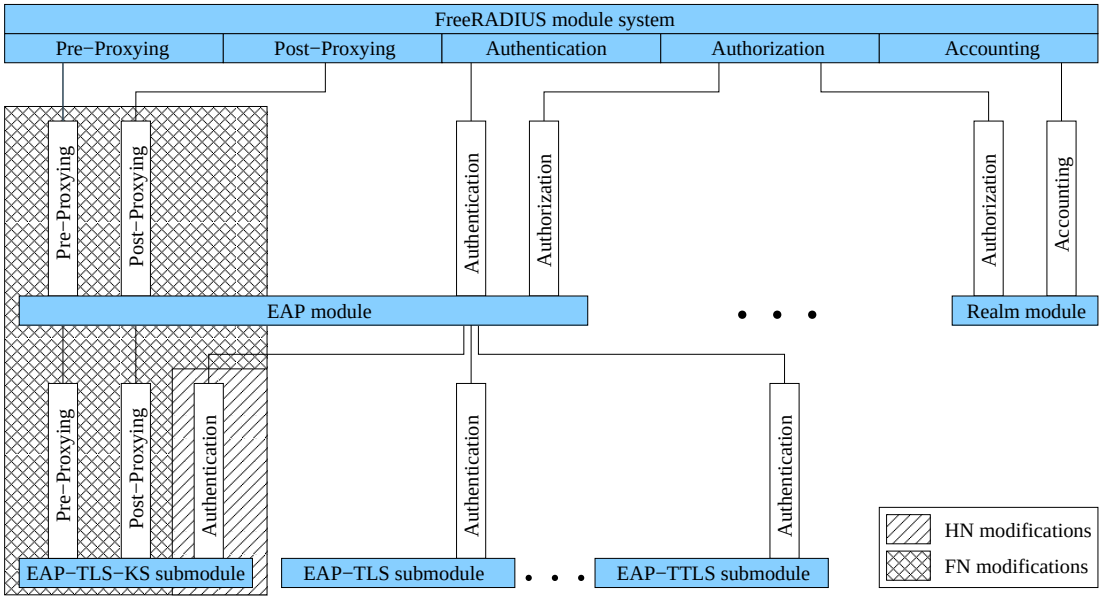

Fig. 4. Overview of the FreeRADIUS module system.

messages. All of these are then forwarded to HN for inclusion in the hash of messages.

At this point, HN verifies the `Client-Certificate-Verify` message. Using a state information callback function registered with OpenSSL's context at HN, the TLS negotation is stopped at this point and an `EAP-TLS-KS-Success` message containing two hash values is returned to FN. The first hash value is used by FN to validate the `Finished` message received from MD. The second hash value is used to generate the `Finished` message to be sent back to MD. The FN then sends back this `Finished` message preceeded by a `Change-Cipher-Spec` message as per the protocol.

Finally, MD sends an acknowledgement of the `Finished` message. FN catches this, but does not forward it to HN. Instead, it generates an `EAP-Success` message and encapsulates it in a RADIUS packet (which includes the keys to be sent to the AP), and sends this to the MD.[4]

This message exchange is significant because this is where EAP-TLS-KS improves performance over EAP-TLS. Since the acknowledgement is not forwarded to the HN and the `EAP-Success` message originates from FN rather than HN, an entire round-trip between HN and FN is eliminated. Due to variables such as certificate size as well as RADIUS and EAP fragmenting, it is not possible to state exactly how many round-trips the authentication will take. For the tests presented in this paper, which used a 1024 bit RSA key and a maximum transmission unit setting of 1500 for the network connections of both FN and HN, there were 13 round-trips between FN and HN for EAP-TLS and only 12 round-trips for EAP-TLS-KS.

[4]As per RFC 2865 [21], since the AP is acting as the Network Access Server (NAS) in the RADIUS exchange, the AP strips the contents of the RADIUS packet. It then forwards just the `EAP-Success` message to MD.

V. Evaluation

A. Testing setup

The testing setup for comparing the two authentication protocols consists of the testbed described in Section IV-A combined with several scripts for automation and processing. Tests are run in blocks. For each block, the hostapd daemon as well as FN and HN RADIUS servers are started manually. A script is then executed to automate the block of test runs.

In each test run, tethereal, the command-line version of ethereal, is started on the machine running both hostapd and the FN RADIUS server. It is configured to collect RADIUS packets sent between FN and HN, EAP packets sent between hostapd and FreeRADIUS over the loopback device, and EAPOL [14] packets sent or received over the wireless interface. It is also configured to exit after the authentication has finished. At this point, wpa_supplicant is run on the MD with a configuration specifying either EAP-TLS or EAP-TLS-KS authentication. Since wpa_supplicant continues to run after authentication to handle re-keying, once tethereal exits, wpa_supplicant is signalled to exit. Two test runs, one for EAP-TLS and one for EAP-TLS-KS, are run in sequence to make a test pair. Each test pair is assigned a sequence number in the block.

After the test block is completed, a shell script runs tethereal over each capture file to produce plaintext output which indicates the time of the `EAP-Start` and `EAP-Success` messages received by MD. The time for the authentication is the difference in these two times. It is stored for further processing.

B. Performance

In total, three test blocks are presented, one to each of the HN locations listed in Section IV-A. Figures 9, 10, and 11 in Appendix II present tests run over a twenty-four hour period.

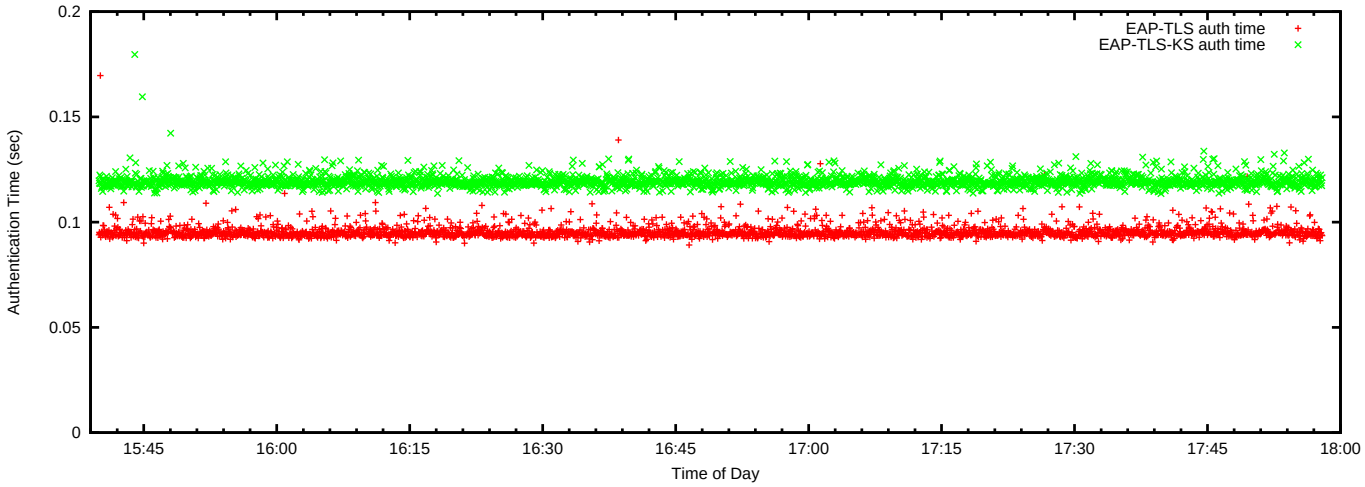

Fig. 5. Local roaming scenario.

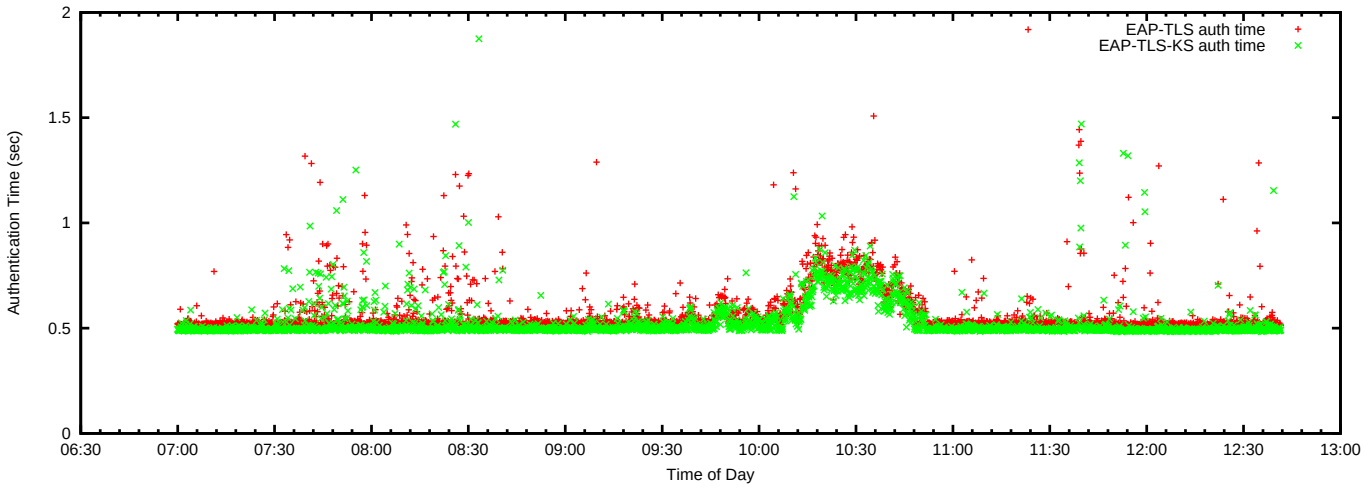

Fig. 6. National roaming scenario.

	EAP-TLS	EAP-TLS-KS
Minimum (sec)	0.089149	0.113571
Maximum (sec)	0.169635	0.179666
Mean (sec)	0.095466	0.119532
Std. Deviation (sec)	0.003395	0.003068

TABLE I
LOCAL ROAMING SCENARIO

	EAP-TLS	EAP-TLS-KS
Minimum (sec)	0.508802	0.481173
Maximum (sec)	2.44112	2.40410
Mean (sec)	0.537552	0.506998
Std. Deviation (sec)	0.074783	0.071419

TABLE II
NATIONAL ROAMING SCENARIO

These tests reveal that a much smaller set of test pairs is sufficient to illustrate the various characteristics present. To display more detail, blocks of 2500 test pairs are analyzed.

Local roaming scenario

Figure 5 illustrates the authentication times for both protocols run to the local HN across campus. Table I presents the corresponding statistics. These tests indicate that with a very high speed connection between FN and HN, the extra computations involved in the EAP-TLS-KS protocol require more time than the additional two messages (one round-trip) between FN and HN that the protocol eliminates (in comparison to EAP-TLS). In the tests, the average time for an EAP-TLS authentication was 0.095466 seconds while the average time for EAP-TLS-KS was 0.119532 seconds.

National roaming scenario

Figure 6 displays the authentication times for both protocols run to the HN in Madison, Wisconsin. Table II presents the

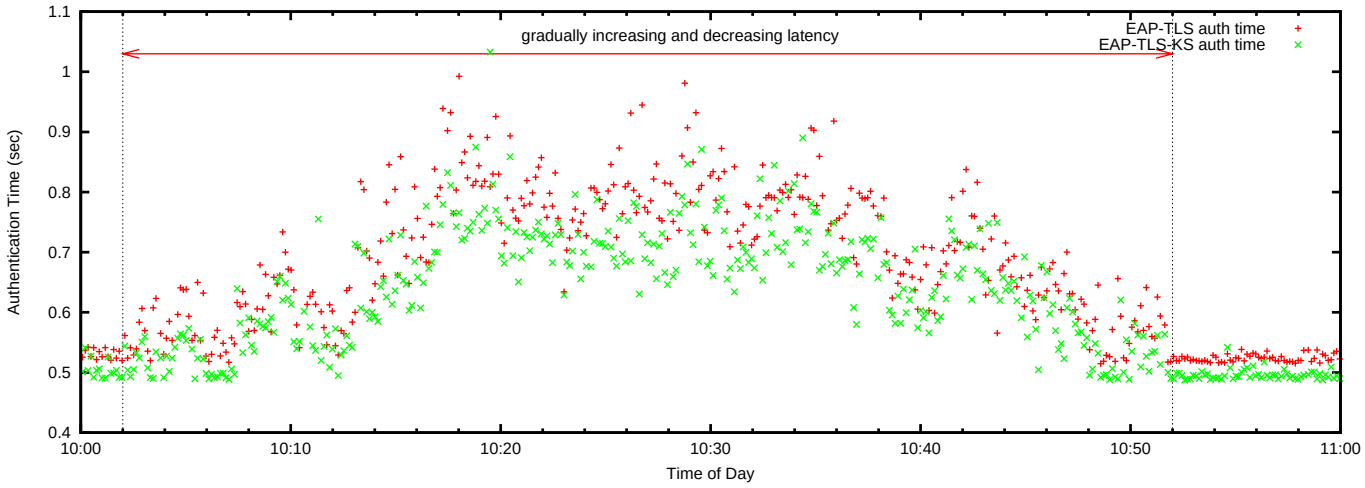

Fig. 7. Period of 10:00am to 11:00am from the National roaming scenario tests shown in Fig. 6.

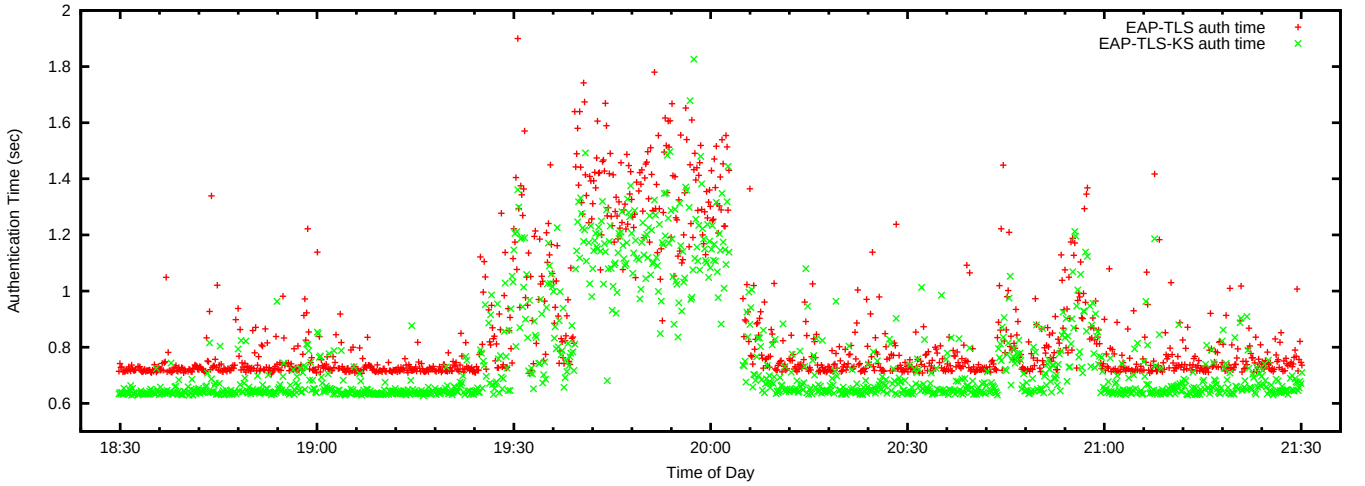

Fig. 8. International roaming scenario.

corresponding statistics. On average, EAP-TLS-KS yielded a 5.68% decrease in authentication time from 0.537552 to 0.506998 seconds. These tests display various other characteristics as well: During periods of stable network latency EAP-TLS-KS is slightly faster than EAP-TLS. During unstable periods there are times when one test is significantly faster than the other. Since the tests were interwoven rather than being executed in a precisely parallel setup, significant[5] network activity could have occurred during only one test of a pair on one of the many links between New Jersey and Wisconsin. However, even in the periods of unstable network latency, EAP-TLS-KS is generally faster than EAP-TLS. Figure 7 shows the period of 10:00am to 11:00am from the national test block. The period between 10:02 and 10:52 is one such period of unstable network latency. Table III presents the statistics for the test pairs in this period. Over this period, EAP-TLS-KS yielded a better than average decrease in authentication time of 8.48% dropping from 0.679742 to 0.622133 seconds.

[5]The definition of significant is very closely tied to the link the traffic is occuring on and its limits.

	EAP-TLS	EAP-TLS-KS
Minimum (sec)	0.514743	0.487355
Maximum (sec)	1.50768	1.12467
Mean (sec)	0.679742	0.622133
Std. Deviation (sec)	0.131806	0.105257

TABLE III
UNSTABLE NETWORK LATENCY PERIOD IN THE NATIONAL ROAMING SCENARIO

International roaming scenario

Figure 8 displays the authentication times for both protocols run to the HN in Germany. Table IV presents the corresponding statistics. Again, various characteristics are visible. Periods of stable and unstable network latency are present, and EAP-TLS-KS continues to outperform EAP-TLS. On average, EAP-

	EAP-TLS	EAP-TLS-KS
Minimum (sec)	0.705426	0.626704
Maximum (sec)	1.89972	1.82637
Mean (sec)	0.865559	0.761858
Std. Deviation (sec)	0.235257	0.194648

TABLE IV
INTERNATIONAL ROAMING SCENARIO

TLS-KS yielded an 11.98% decrease in authentication time from 0.865559 to 0.761858 seconds. Due to the increased network latency to Germany, EAP-TLS-KS yields a greater speed increase in these tests.

Summary and general results

Overall, the results indicate that by trading round-trips for additional cryptographic operations, the DHE-RSA case of EAP-TLS-KS authentication is generally faster than that of EAP-TLS authentication. EAP-TLS-KS also has a smaller variance in authentication time as can be seen in the consistently lower standard deviation. This is directly related to the aforementioned trade. A smaller number of round-trips reduces the possible effect of network instability on authentication time.

While EAP-TLS is faster when there is extremely low network latency, it seems improbable that two WISPs authentication servers would be located within such proximity and with such a high speed connection as to reach this low latency.

C. Other cases of EAP-TLS-KS

Since EAP-TLS-KS was designed to be indistinguishable from EAP-TLS from MD's perspective, it too allows for three different types of certificates to be used as described in Section III. The previous discussion only detailed the DHE-RSA case. One of the other two cases involves use of a certificate for RSA encryption. This case is significantly different from the DHE-RSA case, and due to security concerns, is not suggested for use in new applications. Therefore it will not be discussed further in this paper.

The third case, which uses a certificate for DSS signature verification, is known as the DHE-DSS case. As noted in Section III, this case uses a DH key exchange method in a similar way to the DHE-RSA case. The distributed DSS signature scheme used in this version of EAP-TLS-KS was developed in [18] and is summarized in Appendix I. While the message exchange (but not the message content) in the DHE-DSS case of EAP-TLS-KS is the same as that of the DHE-RSA case, there are several important changes that have to be made in order to implement the DHE-DSS case within the setup described in this paper.

The first change is that FN needs to send $\alpha^{\mathcal{K}_{FN_i}}$ to HN in the `EAP-Identity` message. The next change is much more significant. OpenSSL will generate a valid DSS signature pair (r', s') to be included in the `Server-Key-Exchange` message. This must be replaced at the HN side by the values computed for r and s_{HN}. In addition, the parameters $\alpha^{\mathcal{K}_{HN_i}}$, R_{HN} and R^*_{HN}, will need to be generated and sent in the `Server-Key-Exchange` message. This involves more computation than in the DHE-RSA case, along with significantly enlarging the size of the `Server-Key-Exchange` message.

In addition to the increased computation and message size, the distributed DSS signature generation causes one more problem when implemented in combination with OpenSSL. Since the signature pair (r, s) generated by HN and FN will be different from the pair (r', s') generated by OpenSSL, the hashes of all messages in the negotiations will be incorrect. Therefore, an implementation would have to hash all of these messages.

While these issues do not make implementation of the DHE-DSS case infeasible, they do further complicate implementation. Since there is still an entire round-trip between FN and HN that is eliminated in this case, it seems that the DHE-DSS case of EAP-TLS-KS should show a performance improvement over the same case of EAP-TLS. However, due to increased computation and message size, a higher network latency may be required to make the performance improvement visible.

VI. CONCLUSION AND FUTURE WORK

This paper has shown that in the DHE-RSA case the use of EAP-TLS-KS can decrease the time of the authentication process for a roaming user in a wireless network. If the security benefits of EAP-TLS-KS are factored in, especially the control afforded to both networks, EAP-TLS-KS is an improved authentication mechanism for any roaming scenario.

Future work includes implementing the other cases of EAP-TLS-KS for comparison with EAP-TLS. In addition, blinding for RSA private key operations which use key splits has to be addressed.

REFERENCES

[1] B. Aboba and D. Simon. PPP EAP TLS Authentication protocol. RFC 2716, October 1999.

[2] A. Agarwal, J. Gill, and W. Wang. An experimental study on wireless security protocols over mobile IP networks. In *Proceedings of the IEEE Vehicular Technology Conference (VTC'04-Fall)*, 2004.

[3] A. Agarwal and W. Wang. Measuring performance impact of security protocols in wireless local area networks. `http://www.ece.ncsu.edu/netwis/papers/aw05bn.pdf`, 2005.

[4] K. Bayarou, M. Enzmann, E. Giessler, M. Haisch, B. Hunter, M. Ilyas, S. Rohr, and M. Schneider. Towards certificate-based authentication for future mobile communications. *Wireless Personal Communications*, 29, 2004.

[5] L. Blunk and J. Vollbrecht. PPP Extensible Authentication Protocol (EAP). RFC 2284, March 1998.

[6] D. Boneh and D. Brumley. Remote timing attacks are practical. In *Proceedings of the 12th USENIX Security Symposium*, August 2003.

[7] T. Dierks and C. Allen. The TLS protocol version 1.0. RFC 2246, January 1999.

[8] Ethereal - A Network Protocol Analyzer. `http://www.ethereal.com/`.

[9] The Freeradius Server Project. `http://www.freeradius.org/`.

[10] Gentoo Linux. `http://www.gentoo.org/`.

[11] J. Gu, S. Park, O. Song, L. J., J. Nah, and S. Sohn. Mobile PKI: A PKI-based authentication framework for the next generation mobile communications. In *Proceedings of the Australian Conference on Information Security and Privacy, (ACISP'03)*, 2003.

[12] Host AP driver for Intersil Prism2/2.5/3, hostapd, and WPA Supplicant. `http://hostap.epitest.fi/`.

[13] IEEE. 802.11i - IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements, part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, amendment 6:medium access control (MAC) security enhancements. IEEE Standards Board, July 2004.
[14] IEEE. 802.1X: IEEE standard for local and metropolitian area networks - Port-based network access control, December 2004.
[15] Madwifi - Multiband Atheros Driver for WiFi. `http://www.madwifi.org/`.
[16] Y. Matsunaga, A. S. Merino, T. Suzuki, and R. H. Katz. Secure authentication system for public WLAN roaming. In *Proceedings of the ACM Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH'03)*, September 2003.
[17] A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
[18] U. Meyer, J. Cordasco, and S. Wetzel. An approach to enhance inter-provider roaming through secret-sharing and its application to WLANs. In *Proceedings of the ACM Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH'05)*, September 2005.
[19] P. Prasithsangaree and P. Kishnamurthy. A new authentication mechanism for loosely coupled 3G-WLAN integrated networks. In *Proceedings of the IEEE Vehicular Technology Conference (VTC'04-Spring)*, 2004.
[20] C. Rigney, W. Willats, and P. Calhoun. RADIUS Extensions. RFC 2869, June 2000.
[21] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Authentication Dial in User Services (RADIUS). RFC 2865, June 2000.
[22] L. Salgarelli, M. Buddhikot, J. Garay, S. Patel, and S. Miller. Efficient authentication and key distribution in wireless IP networks. *IEEE Wireless Communications Magazin*, 10(6), 2003.
[23] The Slackware Linux Project. `http://www.slackware.org/`.
[24] D. Stanley, J. Walker, and B. Aboba. Extensible Authentication Protocol (EAP) method requirements for Wireless LANs. RFC 4017, March 2005.

APPENDIX I
DISTRIBUTED DSS SIGNATURE

In [18] a new distributed version of the DSS signature scheme is developed for use in EAP-TLS-KS:

The public roaming key is a DSS signature verification key (p, q, α, y), where p and q are primes, $q|(p-1)$, $\alpha \in \mathbb{Z}_p$, ord$(\alpha) = q, and y = \alpha^a \mod p$. The secret roaming key is a, which is randomly chosen from $\{1, \ldots, q-1\}$.

The signature generation for the hash value $h(m)$ of a message m for non-distributed DSS signatures works as follows: The signer chooses a fresh $k^{-1} \in \{1, \ldots, q-1\}$ for each signature and computes

$$r = \alpha^{k^{-1}} \mod p \mod q$$
$$s = k(h(m) + ar) \mod q.$$

The signature on $h(m)$ then consists of the pair (r, s). For a more detailed description of the DSS signature generation and verification see [17].

For the distributed DSS signature it is necessary to split both the secret key a as well as the ephemeral key k between HN and FN$_i$. HN splits the secret key a for each $i = 1, \ldots, l$ multiplicatively into two parts a_{FN_i} and a_{HN_i}. It distributes a_{FN_i} to FN$_i$ and keeps a copy of each pair of shares (a_{FN_i}, a_{HN_i}).

During signature generation, the ephemeral key k is chosen in a distributed manner. That is, FN$_i$ contributes one part, $\mathcal{K}_{FN_i}$, while HN contributes two parts, $\mathcal{K}_{HN}$ and k_{HN}. $\mathcal{K}_{FN_i}$ is known to FN$_i$ only. $\mathcal{K}_{HN}$ and k_{HN} are known to HN only. $\mathcal{K}_{HN}$ and $\mathcal{K}_{FN_i}$ combine to k_{FN_i} which becomes known to both HN and FN$_i$ during signature generation. The ephemeral key k is the product of k_{HN} and k_{FN_i}. The multiplicative splits of a are generated by first using an additive splitting in the exponent rather than directly splitting it multiplicatively: HN selects $x \in \{1, \ldots, q-1\}$ randomly and chooses $\omega_1, \ldots, \omega_\ell$ randomly in $\mathbb{Z}_{\frac{q-1}{2}}$ with $\omega_i \neq \omega_j$ for $i \neq j$. Then,

$$x_{HN_i} = x + 2\omega_i \mod q - 1$$
$$x_{FN_i} = x + \omega_i \mod q - 1.$$

Thus, $x = -x_{HN_i} + 2x_{FN_i} \mod q - 1$ for all $i = 1, \ldots, \ell$. HN defines $a = \alpha^x \mod p \mod q$ and

$$a_{HN_i} = \alpha^{-x_{HN_i}} \mod p \mod q$$
$$a_{FN_i} = \alpha^{2x_{FN_i}} \mod p \mod q$$

such that $a_{HN_i} \cdot a_{FN_i} = \alpha^{-x_{HN_i}} \cdot \alpha^{2x_{FN_i}} = \alpha^x = a \mod p \mod q$. HN together with any FN$_i$ can now generate a distributed DSS signature during EAP-TLS-KS as follows: FN$_i$ first chooses $\mathcal{K}_{FN_i}$ randomly from $\{0, \ldots q-1\}$ and sends $\alpha^{\mathcal{K}_{FN_i}}$ to HN included in the `EAP-TLS-KS-Start` message. HN then chooses $\mathcal{K}_{HN}, k_{HN}^{-1}, R_{HN}$, and R^*_{HN} randomly in $\mathbb{Z}_q^*$ and computes $k_{FN_i}^{-1} = (\alpha^{\mathcal{K}_{FN_i}})^{\mathcal{K}_{HN}} \mod p \mod q$. Then, HN computes $r = \alpha^{k_{HN}^{-1} \cdot k_{FN_i}^{-1}}$ and

$$\begin{aligned} s_{HN} =& (k_{HN} - R_{HN}) k_{FN_i} \cdot h(m) \\ &+ (k_{HN} \cdot a_{HN_i} - R^*_{HN}) k_{FN_i} \cdot a_{FN_i} \cdot r \\ =& \underbrace{k_{HN} \cdot k_{FN_i} \cdot h(m) + k_{HN} \cdot k_{FN_i} \cdot a_{FN_i} \cdot a_{HN} \cdot r}_{=:s} \\ &- R_{HN} \cdot k_{FN_i} \cdot h(m) - R^*_{HN} \cdot k_{FN_i} \cdot a_{FN_i} \cdot r \end{aligned}$$

HN sends $\alpha^{\mathcal{K}_{HN}}$, r, s_{HN}, R_{HN}, and R^*_{HN} to FN$_i$. FN$_i$ determines $k_{FN_i}^{-1} = (\alpha^{\mathcal{K}_{HN}})^{\mathcal{K}_{FN_i}}$ and

$$s_{FN_i} = k_{FN_i} \cdot R_{HN} \cdot h(m) + k_{FN_i} \cdot R^*_{HN} \cdot a_{FN_i} \cdot r.$$

Now FN$_i$ can compute the signature part s on $h(m)$ as $s = s_{FN_i} + s_{HN}$. The pair (r, s) is now a valid DSS signature on the hash value $h(m)$ with $a = a_{HN_i} \cdot a_{FN_i}$ and $k = k_{FN_i} \cdot k_{HN}$. Thus, it can be verified by MD in the same way as a non-distributed DSS signature with ephemeral key k and secret key a.

APPENDIX II
TWENTY-FOUR HOUR TEST PERIOD GRAPHS

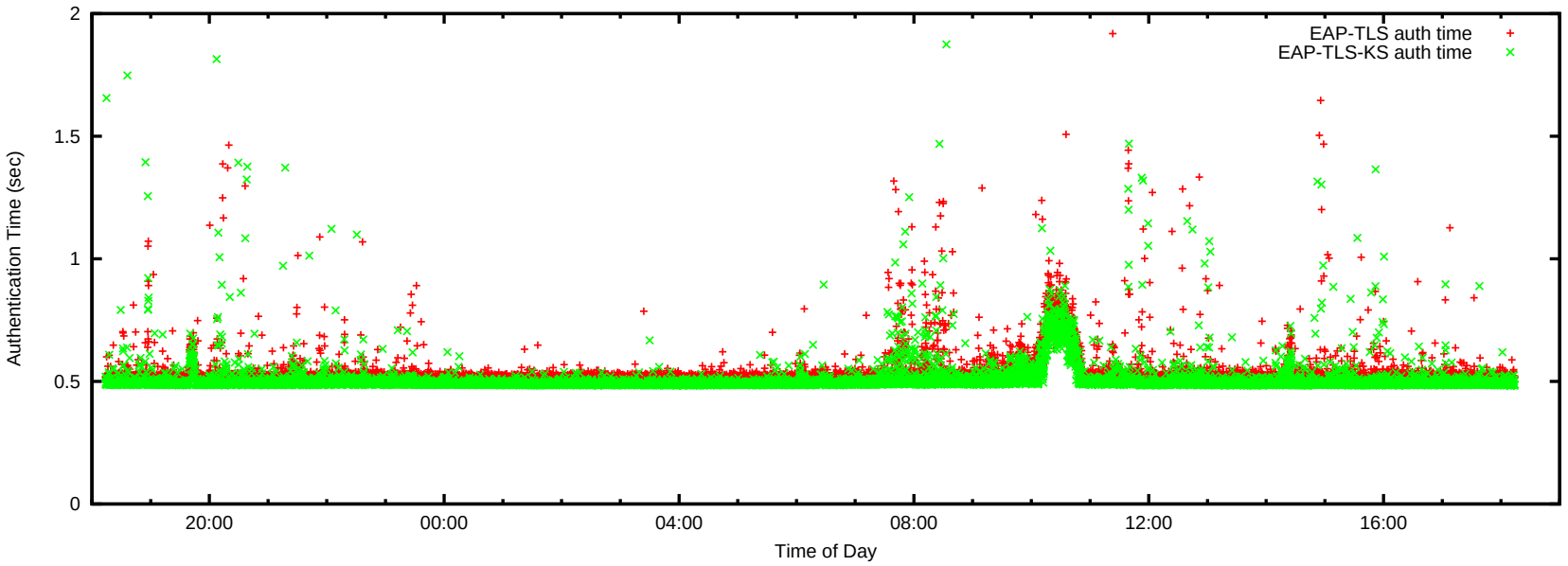

Fig. 10. Twenty-four test period in the national roaming scenario.

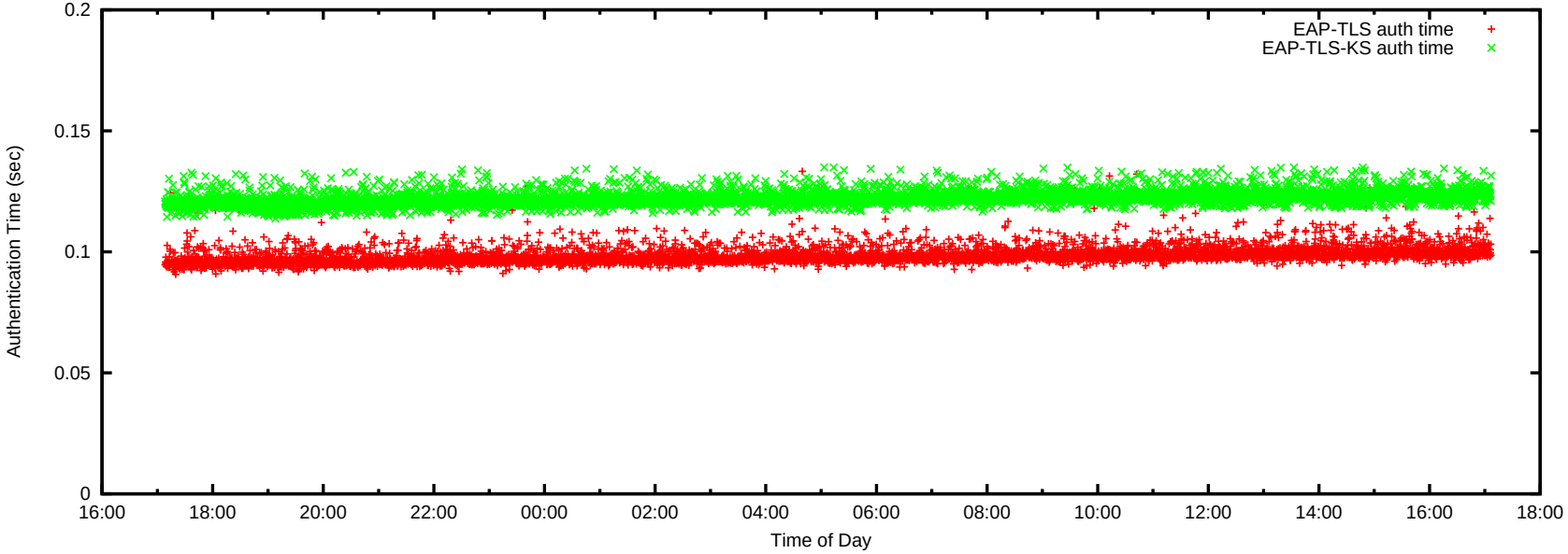

Fig. 9. Twenty-four test period in the local roaming scenario.

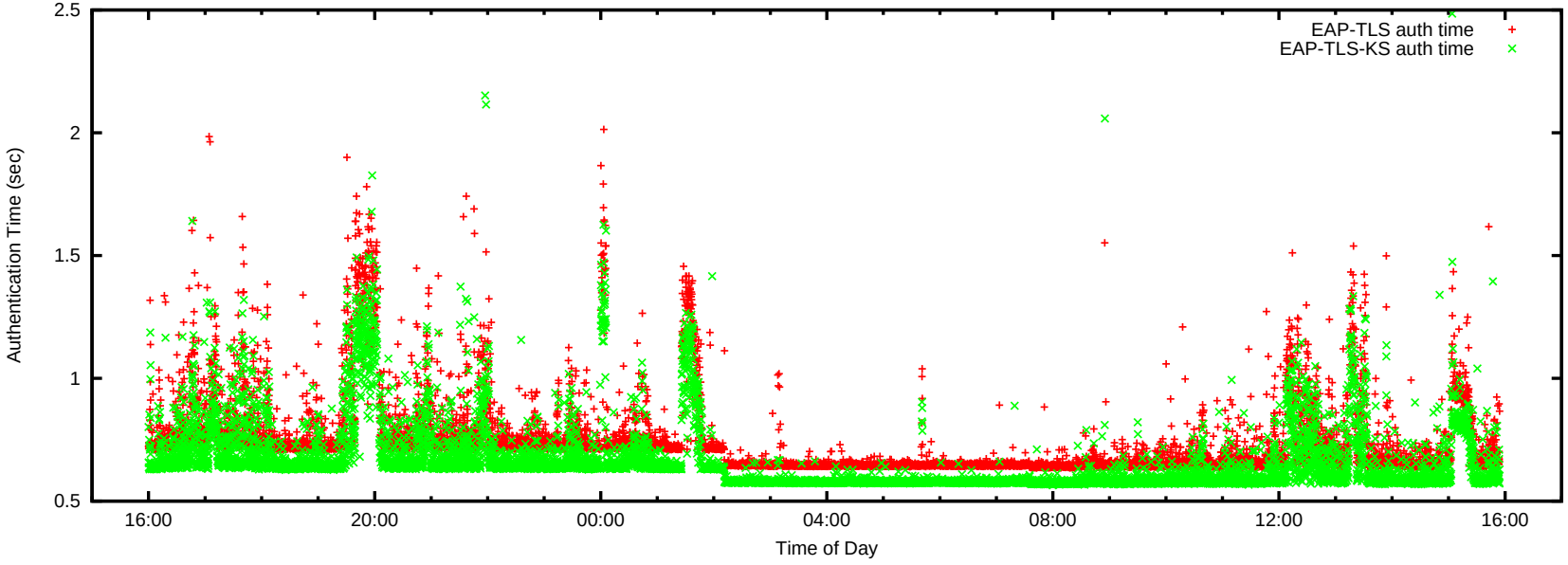

Fig. 11. Twenty-four test period in the international roaming scenario.