An Approach to Enhance Inter-Provider Roaming Through Secret Sharing and its Application to WLANs

Ulrike Meyer[*]
Darmstadt University of Technology
Germany
umeyer@cdc.informatik.tu-darmstadt.de

Jared Cordasco and Susanne Wetzel
Stevens Institute of Technology
USA
{jcordasc,swetzel}@cs.stevens.edu

ABSTRACT

In this paper, we show how secret sharing can be used to address a number of shortcomings in state-of-the-art public-key-based inter-provider roaming. In particular, the new concept does not require costly operations for certificate validation by the mobile device. It furthermore eliminates the need for a secure channel between providers upon roaming. We demonstrate the new approach by introducing a new protocol, EAP-TLS-KS, for roaming between 802.11i-protected WLANs. In addition, we show that the properties of EAP-TLS-KS allow for an efficient integration of a micropayment scheme.

Categories and Subject Descriptors

C.2.0 [**Computer-Communications Networks**]: Security and Protection; C.2.1 [**Network Architecture and Design**]: Wireless Communications

General Terms

Security

Keywords

802.11i, EAP-TLS-KS, inter-provider roaming, micropayment scheme, secret sharing, PKI, distributed DSS, WLAN

1. INTRODUCTION

As WLAN hotspots become more widely available in airports, train stations, coffee shops and hotels, there is an increasing need for easy to use authentication protocols which enable roaming between different Wireless Internet Service Providers (WISPs). Most WISPs currently use the web based Universal Access Method (UAM) for authentication—a method which is also recommended as the best current practice for inter-provider roaming by the Wi-Fi Alliance [5]. However, UAM is known to be vulnerable to many different attacks such as impersonation of an access point, dictionary attacks, and service theft by means of address spoofing [46]. The new standard 802.11i [25] was put forward to address these problems. It uses MAC-layer encryption between mobile devices and access points and is thus secure against service theft by means of address spoofing. Furthermore, it requires mutual authentication between a Mobile Device (MD) and a network. The authentication methods supported by the standard either use public-key certificates or are based on other types of credentials.

Public-key-based methods in principle have the advantage that a Foreign Network (FN) and MD can authenticate each other without the involvement of MD's Home Network (HN). However, most roaming scenarios—in particular commercial ones—require that every instance of roaming be controlled by HN. Therefore, most public-key-based methods and all non-public-key-based methods suggested for roaming to date require that the mutual authentication involves HN. That is, HN authenticates MD and assures FN of MD's authorization to roam to FN. Similarly, HN authenticates FN and assures MD of FN's authorization to offer service to MD.

Furthermore, in authentication methods that require HN's interaction, the cryptographic keys used for MAC-layer protection (between MD and FN) following a successful authentication are typically generated by HN. This requires HN to establish a secure channel to FN in order to allow for a secure transfer of these cryptographic keys.

Another shortcoming which is common to all public-key-based authentication methods for roaming users is that MD must check the validity and revocation status of certificates during network authentication, i.e., before actually having network access. Recent work (e.g., [10]) addresses this problem by delegating certificate chain discovery and validation to a trusted authority.

In this paper, we address the shortcomings of state-of-the-art public-key-based roaming methods by introducing suitable secret sharing techniques.

Contributions:

First, instead of issuing an individual PKI-certificate for each FN, we suggest for HN to use only one roaming certificate. HN shares the secret key corresponding to the roaming certificate with each one of its roaming partners by means

[*]The research was done while the author was at Stevens Institute of Technology. The visit was partly funded by the German Academic Exchange Service (DAAD) and the German Research Foundation (DFG).

of a $(2, 2)$ secret sharing scheme. Upon roaming to FN, MD authenticates FN based on the pre-installed roaming certificate of its HN. The key splitting guarantees that MD cannot authenticate FN without HN's participation.

Second, we show that our new key splitting approach allows for efficient certificate handling. In particular, MD does not need to validate any certificates upon roaming as HN's certificate is pre-installed on MD.

Third, we show that the use of secret sharing eliminates the need for a secure channel between FN and HN upon public-key-based roaming of MD. This is due to the fact that the splitting of the secret key allows FN to derive any necessary keying material itself.

Furthermore, we present a new protocol EAP-TLS-KS which implements the new concept based on EAP-TLS. The new protocol can be used as an EAP-Method within 802.11i and as such enhances WLAN inter-provider roaming. EAP-TLS-KS is designed such that it differs from the original EAP-TLS protocol only on the server side. In particular, it is the public-key operations which are performed by HN in EAP-TLS—such as decryptions and signatures—that are split between HN and FN in EAP-TLS-KS. We specify three protocol variants in order to support all types of EAP-TLS certificates. The first and second variants use conventional distributed RSA operations. For the third variant, we present a new distributed DSS signature scheme. This new scheme can generally be used in applications that exhibit an asymmetry in signing capabilities of the individual parties. That is, while one party (in our case HN) can generate valid signatures on its own, the second party (in our case FN) requires the other party's cooperation in order to generate a valid signature. We also show that the new EAP-TLS-KS protocol has a performance advantage over EAP-TLS. While the original EAP-TLS protocol needs four round-trip message exchanges between HN and FN, the new protocol requires only two.

Aside from addressing the security problems of current roaming solutions we also show how key splitting can be used to efficiently support the fine grained billing of a micropayment scheme. In our key splitting, HN keeps an individual share of the secret roaming key for each FN. An authentication based on the roaming key can thus not only prove FN's authorization by HN to MD but also prove FN's identity to MD. This allows MD to issue micropayments for a particular FN. Exploiting this fact in the context of EAP-TLS-KS, we present a modified version of the micropayment scheme introduced in [14] and show that its expensive initialization phase can efficiently be integrated into EAP-TLS-KS: the integration adds only one message to our new protocol. We also show that the original micropayment scheme suffers from a malicious service provider attack. Our modified protocol can be protected against this attack due to MD's knowledge of FN's identity.

Outline: In Section 2 we give an overview of the security architecture of 802.11i. This is followed by an overview of EAP-TLS in Section 3. In Section 4 we first introduce the new concept of roaming with key splitting and then focus on its application to WLAN. In particular, we introduce a new method for distributed DSS signatures in Section 4.2.3. This is followed by a detailed discussion of the EAP-TLS-KS protocol including a security analysis. Our accounting mechanism is described in Section 5. We close the paper by summarizing related work in Section 6.

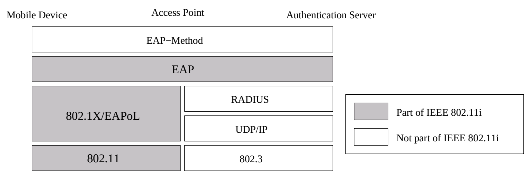

Figure 1: Protocol Architecture in 802.11i

2. OVERVIEW OF 802.11i

In June 2004, 802.11i was adopted as the new security standard for the wireless LAN technology 802.11 [25]. It replaces the original security architecture WEP (Wired Equivalent Privacy) which was shown to be insecure in many respects. 802.11i offers access control via mutual authentication between MD and the network. In addition, it protects the confidentiality and integrity of the air interface between MD and an Access Point (AP).

2.1 Authentication

The new standard, 802.11i, supports two different authentication and key agreement methods: one that is based on a pre-shared key and one that is based on 802.1X [24]. In the following we only refer to the latter.

The 802.1X-based authentication is implemented as a protocol between MD, the AP it associates with, and an Authentication Server (AS) which controls the network access for one or more APs. The MD first associates with an AP within its range using WEP's open authentication. The association only enables MD to exchange authentication data with AS. Any other traffic is blocked until the authentication is completed successfully. The standard does not specify any particular authentication protocol to be used between MD and AS. Instead, it specifies a WLAN-adapted implementation of the Extensible Authentication Protocol (EAP) [11] on top of which different authentication methods can be used.

Figure 1 describes the general protocol architecture between MD, AP, and AS. EAP-Method stands for the actual authentication mechanism used. EAP-Methods defined to date include EAP-TLS [2], EAP-TTLS [15], and EAP-SIM [22]. It is important to note that only key generating EAP-Methods can be used in connection with 802.11i. EAP itself is the end-to-end transport protocol for the EAP-Method between MD and AS. EAPoL transports EAP over 802.x LANs and implements a port-based access control. Each association of MD with an AP creates a pair of IEEE 802.1X-controlled ports. Both sides implement a port blocking that blocks all traffic until the 802.1X authentication procedure completes successfully. RADIUS [40] can be used to transport EAP over IP to establish an authenticated channel between AP and AS as well as to securely transport the generated key from AS to AP. The use of RADIUS is not required but suggested in the standard.

2.2 Key Generation

If the 802.1X-based authentication is used, MD and AS generate a secret Pairwise Master Key (PMK) using an EAP-Method. After generating the PMK, AS transfers PMK to the AP that MD is associated with.

Both AP and MD use the EAPoL-Handshake to generate a Pairwise Transient Key (PTK). PTK is derived from PMK, the MAC addresses of MD and AP, as well as two

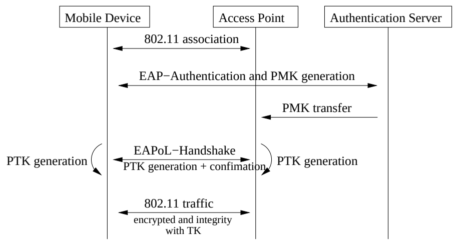

Figure 2: Overview of 802.11i

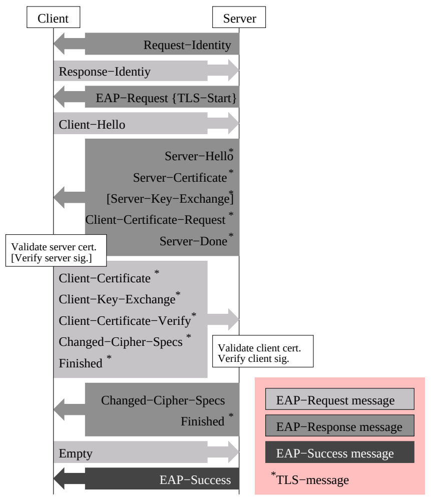

Figure 3: Overview of the EAP-TLS Protocol

nonces generated by and exchanged between the two parties. PTK consists of three parts. The first one is used for key confirmation in the EAPoL-Handshake, the second part is used for encrypted transfer of the Group Transient Key (GTK), used for broadcast traffic from AP to all associated MDs, and the last part is used as the Temporal Key (TK) for encryption and integrity protection of the subsequent 802.11 traffic. Figure 2 provides an overview of the 802.1X-based authentication and key agreement.

3. OVERVIEW OF EAP-TLS

EAP-TLS is an EAP-Method defined in RFC 2716 [2] based on TLS [16]. It supports either mutual public-key certificate-based authentication or server authentication only. If EAP-TLS is used in 802.11i for server authentication only, another authentication method must be combined with EAP-TLS to implement client authentication. In the following we describe the use of EAP-TLS with mutual certificate-based authentication. Figure 3 gives an overview of the EAP-TLS protocol and shows the encapsulation of TLS in EAP messages: After agreeing upon the use of EAP, AS sends the client (in our case MD) an `EAP-Request` message requesting the client's identity. The client answers with an `EAP-Response` message including its identity. AS then sends the `TLS-Start` message in an `EAP-Request` message and the TLS-Handshake begins:

The client sends the `TLS-Client-Hello` message in an `EAP-Response` message to the server. `Client-Hello` includes a random number Client.RAND that guarantees the freshness of the resulting keys to the client. The server answers with an `EAP-Request` message including the TLS messages `Server-Hello`, `Server-Certificate`, `Client-Certificate-Request` and `Server-Done`, and optionally the `Server-Key-Exchange` message. `Server-Hello` includes a random number Server.RAND that guarantees key freshness to the server. The certificate of the server is of one of the following three kinds:

1. A certificate including a public RSA key usable for encryption and signed by a CA with an RSA signature key. (RSA)

2. A certificate including a public RSA key usable for RSA signature verification, signed by a CA with an RSA signature key. (DHE-RSA)

3. A certificate including a public DSS key usable for DSS signature verification, signed by a CA with a DSS signature key. (DHE-DSS)

EAP-TLS supports two methods for generating keying material. One is RSA encryption based (RSA case) and the other is based on a Diffie-Hellman key exchange (DHE case). In the RSA case, the server uses a certificate of type RSA and no `Server-Key-Exchange` is sent. In the DHE case, the server uses a certificate of type DHE-RSA or DHE-DSS and `Server-Certificate` is followed by `Server-Key-Exchange`. This message includes the server's public DH value for this protocol instance. The hash value of the server's public DH value concatenated with Client.RAND and Server.RAND is signed with the server's RSA or DSS signature key and included in `Server-Key-Exchange`.

The client answers with `EAP-Response` including the TLS messages `Client-Certificate`, ..., `Finished`. `Client-Certificate` is a DHE-RSA or a DHE-DSS certificate, depending on what type of certificate the server requests. `Client-Key-Exchange` is different for the RSA case and the DHE case:

1. In the RSA case, `Client-Key-Exchange` includes a random number Sec.RAND generated by the client and encrypted with the server's public key.

2. In the DHE case, `Client-Key-Exchange` message includes the client's public DH value.

In order to prove its identity to the server, the client's response includes `Client-Certificate-Verify`. This message contains a hash value of all messages sent and received so far starting from `Client-Hello` up to and including `Client-Key-Exchange` and is signed using the client's signature key. The same `EAP-Response` message also includes `Change-Cipher-Specs` and `Finished`. With the former, the client 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 the client uses the same cipher-suite and keys as the server.

The server answers with the `Change-Cipher-Specs` and the `Finished` messages. By verifying the correct encryption of the `Finished` message, the client obtains a proof of the server's identity since only the server can generate the correct session key. The client 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 the server to the client.

Figure 4 details the RSA case. The server certificate includes a public RSA encryption key. `Server-KeyExchange`

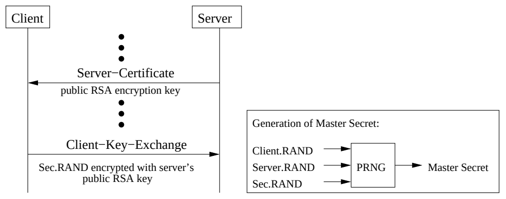
Client
Server
Server−Certificate
public RSA encryption key
Client−Key−Exchange
Sec.RAND encrypted with server's public RSA key
Generation of Master Secret:
Client.RAND
Server.RAND
Sec.RAND
PRNG
Master Secret

Figure 4: EAP-TLS with RSA

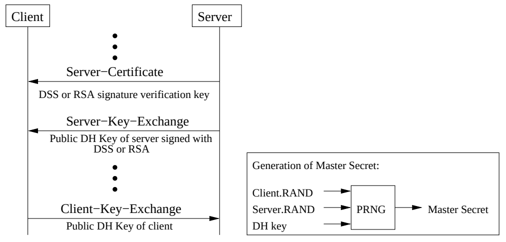

Figure 5: EAP-TLS with DHE

is not sent. `Client-Key-Exchange` consists of a random number Sec.RAND encrypted with the server's public RSA key. Server and client generate the master secret by using Client.RAND, Server.RAND and Sec.RAND as input to a Pseudo-Random Number Generator (PRNG). Figure 5 details the DHE case. `Server-Certificate` includes a public RSA or DSS signature verification key. `Server-Key-Exchange` includes the public DH value of the server and is signed with the server's private RSA or DSS key. `Client-Key-Exchange` includes the public DH value of the client. The client computes a DH key from the public DH value of the server and its secret DH value. The server computes the DH key from the public DH value of the client and its own secret DH value. Both compute the master secret by using the Client.RAND, the Server.RAND and the common DH key as input to a PRNG.

4. ROAMING WITH KEY SPLITTING

4.1 General Concept

In state-of-the-art public-key-based roaming, MD authenticates FN based on FN's individual public-key certificate. The key idea of using secret sharing in this setting is to replace the individual certificates for FNs by means of a suitable key splitting between HN and FNs in combination with issuing a certificate for HN only. In detail, this works as follows:

Every HN is issued one roaming certificate. Assuming HN has a pairwise roaming agreement with l foreign networks $\mathrm{FN}_1, \dots \mathrm{FN}_l$, HN splits its secret roaming key $\mathfrak{R}$ into l different pairs of shares $(\mathfrak{R}_{\mathrm{HN}_i}, \mathfrak{R}_{\mathrm{FN}_i})$ by means of individual $(2,2)$ secret sharing schemes with $\mathfrak{R}_{\mathrm{HN}_i} \neq \mathfrak{R}_{\mathrm{HN}_j}$ and $\mathfrak{R}_{\mathrm{FN}_i} \neq \mathfrak{R}_{\mathrm{FN}_j}$ for $i \neq j$. HN then distributes $\mathfrak{R}_{\mathrm{FN}_i}$ to FN_i.[1] Unlike in other secret sharing applications, in our approach HN keeps copies of $\mathfrak{R}_{\mathrm{HN}_i}$ as well as the secret roaming key $\mathfrak{R}$. This not only allows HN to use the secret roaming key in case MD wants to access HN directly but also enables HN to issue suitable shares to new roaming partners. By construction, $\mathfrak{R}$ can be recovered from a collection of shares, if and only if this collection includes a pair $(\mathfrak{R}_{\mathrm{HN}_i}, \mathfrak{R}_{\mathrm{FN}_i})$ for some $i \in \{1, \dots, l\}$. In particular, $\mathfrak{R}$ cannot be reconstructed from any pair $(\mathfrak{R}_{\mathrm{HN}_i}, \mathfrak{R}_{\mathrm{FN}_j})$ (with $i \neq j$) or any collection of shares of foreign networks only. Constructing key pairs with $(\mathfrak{R}_{\mathrm{HN}_i}, \mathfrak{R}_{\mathrm{FN}_i}) \neq (\mathfrak{R}_{\mathrm{HN}_j}, \mathfrak{R}_{\mathrm{FN}_j})$ for $i \neq j$ is necessary in order to allow for unique identification of FN_i upon successful authentication.

Authenticating FN by MD using public-key certificates generally involves related operations by both FN and HN such as decryptions or generating signatures. By introducing the mechanism of key splitting, these operations need to be adapted accordingly. In particular, these operations are now split between HN and FN using distributed decryption or distributed signature generation.

Upon roaming to FN in this new framework, MD now always uses the pre-installed certificate of its HN regardless of FN's identity. Consequently, MD does not have to validate any certificate. In particular, HN's participation in a successful authentication automatically confirms the use of a valid roaming key.[2] Aside from simplifying the handling of certificates considerably, the new key splitting approach may entail additional advantages over state-of-the-art solutions. This potentially includes a reduction of the number of round-trip message exchanges between MD and HN. Similarly, it may eliminate the need for a secure channel between FN and HN which, for example, is used to transfer a master key from HN to FN. This is due to the fact that the key splitting allows FN to use its share to derive the master key from information secured by HN's share.

In the remainder of this section, we introduce the EAP-TLS-KS protocol which implements the new key splitting approach based on EAP-TLS, thus enhancing WLAN inter-provider roaming.

4.2 EAP-TLS with Key Splitting

Applying the new concept of key splitting to EAP-TLS, each HN is issued a roaming certificate, that includes a public RSA encryption key, a public RSA signature verification key, or a public DSS signature verification key. HN splits and distributes the secret roaming key as described in Section 4.1. Each MD stores the roaming certificate of its HN.

Upon roaming to FN, FN and MD initiate an EAP-TLS-KS authentication. MD acts like a regular client in the EAP-TLS protocol as the EAP-TLS-KS protocol differs from EAP-TLS only on the server side. Depending on the type of roaming certificate, either RSA decryption, RSA signature generation or DSS signature generation is split between HN and FN using their respective shares of the secret roaming key. In the following we detail the distributed schemes as well as the key splitting for an RSA encryption key, an RSA signature key, and a DSS signature verification key as the public roaming key.

4.2.1 Distributed RSA Decryption

The public roaming key is a pair (n, e) of an RSA modulus $n = pq$ with p, q prime and an RSA encryption key e with $\gcd(e, \varphi(n)) = 1$. The secret roaming key d is the inverse of e modulo $\varphi(n)$: $ed = 1 \mod \varphi(n)$. Let l be

[1]In standard secret sharing notation, this corresponds to implementing the access structure $\Gamma = \{\{\mathrm{HN}\}, \{\mathrm{HN}_1, \mathrm{FN}_1\}, \dots, \{\mathrm{HN}_l, \mathrm{FN}_l\}\}$ as an iterative threshold scheme of type $(1, l)[(2,2), \dots, (2,2)]$. Unlike in the conventional secret sharing setting, in our approach the HN_i $(i = 1, \dots, l)$, in fact, do not represent distinct share holders but the respective shares are all held by HN.

[2]In order to avoid an impersonation attack, it is mandatory for HN to immediately notify all of its MDs of the revocation of the current roaming certificate due to compromise and distribute a new one.

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 $i = 1, \ldots, l$ implementing the access structure Γ in the following way: HN randomly chooses $\omega_1, \ldots, \omega_l \in \mathbb{Z}_{\frac{\varphi(n)}{2}}$ such that $\omega_i \neq \omega_j$ for $i \neq j$. Then,

$$\begin{aligned} d_{HN_i} &= d + 2\omega_i \mod \varphi(n), \quad i = 1, \ldots, l \\ d_{FN_i} &= d + \omega_i \mod \varphi(n), \quad i = 1, \ldots, l. \end{aligned}$$

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

Since $\omega_i \neq \omega_j$, each FN gets a different share. Since additionally $0 \leq \omega_i \leq \frac{\varphi(n)}{2}$, it holds that $2\omega_i \neq 2\omega_j$ for $i \neq j$. Thus, HN keeps a different share for each FN. The pair of shares (d_{HN_i}, d_{FN_i}) for a foreign network FN_i thus uniquely carries the identity of FN_i.

HN and any single FN_i can now decrypt a message m encrypted to $c = m^e$ with the public encryption key e in a distributed way: HN first computes $c^{-d_{HN_i}}$ and sends the result to FN_i. FN_i then computes $c^{-d_{HN_i}} c^{2d_{FN_i}} = c^{-d_{HN_i} + 2d_{FN_i}} = c^d = m$.

Security Analysis: No FN can decrypt any message encrypted with e on its own as the knowledge of its share does not reveal any information about the decryption key d. Furthermore, by construction no pair or any larger coalition of FNs learn anything about the secret key from combining their key shares.[3]

An attacker intercepting the message $c^{-d_{HN_i}}$ sent from HN to FN_i, does not obtain any information on the plaintext m. Otherwise, the attacker would be able to break an RSA encryption with the public key $-ed_{HN_i}$ which contradicts the RSA assumption. Likewise, an attacker does not gain any information on d_{HN_i} from his knowledge of $c^{-d_{HN_i}}$. For a more formal security analysis of additional properties of this distributed RSA decryption scheme we refer to [12].

4.2.2 Distributed RSA Signatures

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 exactly as in the RSA encryption-based case previously described. HN, together with any one of the foreign networks FN_i, can sign the hash value $h(m)$ on a message m in the following distributed way: HN half signs $h(m)$ 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 $h(m) = s^e \mod n$.

Security Analysis: By construction, knowing only its share does not allow FN to sign a hash value by itself. Furthermore, no pair or larger coalition of FNs learn anything about the secret key by pooling their key shares.

An attacker intercepting the half signed hash value $h(m)^{-d_{HN_i}}$ sent from HN to FN_i cannot complete the signature without knowledge of d_{FN_i}. If an attacker could generate a valid signature $h(m)^d$ on $h(m)$ from $h(m)^{-d_{HN_i}}$, he could compute $h(m)^{2d_{FN_i}} = h(m)^{d - d_{HN_i}}$ and thus generate valid RSA signatures for a secret key d_{FN_i}. Thus, the above distributed signature scheme is as secure as the original RSA signature. For a formal analysis of additional security properties we refer to [30].

4.2.3 New Distributed DSS Signatures

The public roaming key is a DSS signature verification key (p, q, α, y), where p and q are primes, $q | (p-1)$, $\alpha \in \mathbb{Z}_p$, $\mathrm{ord}(\alpha) = q, y = \alpha^a \mod p$. 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 signatures works as follows: The signer chooses a fresh $k^{-1} \in \{1, \ldots, q-1\}$ for each signature and computes

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

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 [34].

For a 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. In our new signature scheme, 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}.[4]

Without loss of generality, the multiplicative splits of a are generated by first using an additive splitting in the exponent rather than directly splitting it multiplicatively:[5] HN selects $x \in \{1, \ldots, q-1\}$ randomly and chooses $\omega_1, \ldots, \omega_l$ randomly in $\mathbb{Z}_{\frac{q-1}{2}}$ with $\omega_i \neq \omega_j$ for $i \neq j$. Then,

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

Thus, $x = -x_{HN_i} + 2x_{FN_i} \mod q-1$ for all $i = 1, \ldots, l$. Obviously, all shares are different: $x_{HN_i} \neq x_{HN_j}$ for $i \neq j$ and $x_{FN_i} \neq x_{FN_j}$ for $i \neq j$. HN defines $a = \alpha^x \mod p \mod q$ and

$$\begin{aligned} a_{HN_i} &= \alpha^{-x_{HN_i}} \mod p \mod q, \ i = 1, \ldots, l \\ a_{FN_i} &= \alpha^{2x_{FN_i}} \mod p \mod q, \ i = 1, \ldots, l \end{aligned}$$

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 can now generate a distributed DSS signature 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. HN then chooses $\mathcal{K}_{HN}, k_{HN}^{-1}, R_{HN}$ and R_{HN}^* randomly in $\mathbb{Z}_q^*$ and computes $k_{FN_i}^{-1} = (\alpha^{\mathcal{K}_{FN_i}})^{\mathcal{K}_{HN}} \mod p \mod q$. HN ensures that $k = k_{FN_i} k_{HN_i} \mod q$ has not yet been used with the same secret key a before. Then, HN computes

[3]The foreign networks learn, that their shares themselves are not the secret. This reduces the number of possible values for d from $\varphi(n)$ to $\varphi(n) - k$.

[4]Instead of generating k_{FN_i} as a combination of $\mathcal{K}_{HN}$ and $\mathcal{K}_{FN_i}$, FN can alternatively generate k_{FN_i} on its own and send it to HN. In this case, however, a secure channel between HN and FN_i is needed.

[5]It can easily be checked that both methods are equivalent. Nevertheless, the former allows for a simpler argument in that all splits are different.

$r = \alpha^{k_{HN}^{-1} \cdot k_{FN_i}^{-1}}$ 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.

Security Analysis: By construction, FN cannot generate a valid signature on its own as its share does not provide any information on the key a.

An attacker cannot generate s_{FN_i} without knowledge of k_{FN_i} and a_{FN_i}. This is due to the fact that s_{FN_i} is indeed a DSS signature on $R_{HN} \cdot h(m)$ with ephemeral key k_{FN_i} and long term key $a_{FN_i} \cdot R_{HN}^*$. If DSS is secure against existential forgery, then it is not possible to generate s_{FN_i} without knowledge of k_{FN_i} and $a_{FN_i} \cdot R_{HN}^*$. This is equivalent to the knowledge of k_{FN_i} and a_{FN_i} as R_{HN}^* is public.

Two or more collaborating FNs cannot generate a valid signature as they cannot reconstruct the secret key a from their shares.

From intercepting $\alpha^{\mathcal{K}_{HN}}$, r, s_{HN}, R_{HN}, R_{HN}^* and (s, r) an attacker cannot derive any information on a, a_{HN_i} or a_{FN_i}.

HN chooses its contribution to k without revealing it to FN. HN makes sure that no value of k is used twice to generate a signature. FN or any attacker that interferes with the DH exchange used to exchange k_{FN_i} thus can not force the same k to be used twice.[6]

Unlike in the two RSA cases discussed previously, in the DSS case HN uses its knowledge of the shares of the FNs during the signature generation process. As discussed earlier, it is HN that should control authentication as it will be the entity responsible for accounting. Nevertheless, it is important to note that the discussed distributed version of DSS is restricted to areas of application where one of the signers can sign on its own, while the other will need the cooperation of the first party to generate signatures.

Section 6 will provide a detailed discussion on how our distributed DSS signature scheme differs from previous work in the area.

4.3 The Protocol

4.3.1 *RSA Encryption-Based Key Generation*

Figure 6 describes the EAP-TLS-KS protocol in the case where an RSA encryption key is used as the public roaming key and the key generation is based on RSA encryption. The protocol starts with the regular `EAP-Request-Identity` and `EAP-Response-Identity` messages, and the `EAP-TLS-Start` message between FN's authentication server and MD. After receiving MD's identity, FN sends an `EAP-TLS-KS-Start` message to HN. This message includes the client's EAP-ID as well as the identity of FN. All of the following EAP-TLS messages, starting with `Client-Hello` and ending with `Client-Certificate-Verify`, are forwarded between MD and HN by FN.

[6]It is well known that in DSS, using the same ephemeral key k twice with the same secret key a reveals a [34].

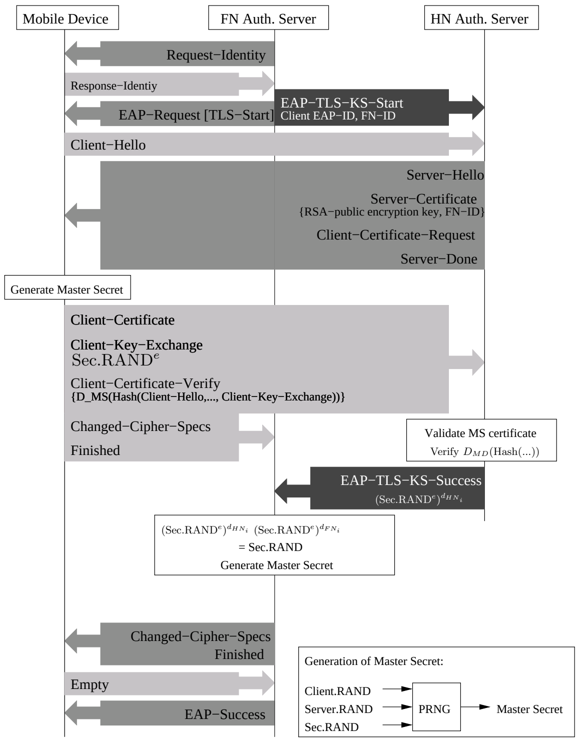

Figure 6: EAP-TLS-KS with RSA

In order to acknowledge FN's identity to MD, HN includes FN's identity in any of the TLS messages it sends to MD. In order to avoid any changes to the EAP-TLS implementation for MD, we integrate FN's identity (FN-ID) as an attribute into the roaming certificate. MD can thus store the roaming certificate and refer to FN's identity at any time. Upon receipt of the `Server-Certificate` message (which includes the roaming certificate), MD checks that the roaming key in the certificate matches the pre-installed key.

If they are equal, then MD chooses a random number Sec.RAND and encrypts it under the public roaming key e. MD computes the hash value of all messages from the `Client-Hello` up to the `Client-Key-Exchange` messages, signs this hash value and includes it in the `Client-Certificate-Verify` message. MD then sends the `Client-Certificate`, `Client-Key-Exchange`, `Client-Certificate-Verify`, `Change-Cipher-Specs` and the `Finished` messages to FN.

FN forwards the messages `Client-Certificate`, `Client-Key-Exchange` and `Client-Certificate-Verify` to HN. HN verifies MD's certificate and its revocation status as well as the signature on the `Client-Certificate-Verify` message. The correctness of the signature proves to HN that MD is in possession of the secret key corresponding to the public key in MD's certificate. It furthermore proves that none of the messages exchanged between MD and HN so far have been altered in any way. In particular, this proves that MD received the same FN-ID as an attribute in the roaming certificate that HN sent. Thus, HN knows that both HN and MD associate the same identity with FN.

If the signature verification is successful, HN sends the half decrypted random number $(\text{Sec.RAND}^e)^{-d_{HN_i}}$ to FN. The receipt of this message assures FN that HN has successfully authenticated MD and thus authenticates MD indirectly to FN. FN fully decrypts Sec.RAND by computing $(\text{Sec.RAND}^e)^{-d_{HN_i}} \cdot (\text{Sec.RAND}^e)^{2d_{FN_i}} = \text{Sec.RAND}$. FN can now compute the secret master key $\text{PMK} = \text{PRNG}(\text{Client.RAND}, \text{Server.RND}, \text{Sec.RAND})$.

The `Change-Cipher-Spec` message indicates that the sending party will now switch to encryption mode. The `Finished` messages exchanged between FN and MD are encrypted with the secret master key PMK. By verifying that the `Finished` message it received is correctly encrypted, MD is assured that FN was able to generate the correct key. MD is furthermore assured that HN participated in the authentication and that the identity FN claimed to HN is correct and corresponds to the one included as an attribute in the roaming certificate.

4.3.2 DHE-RSA Case

Figure 7 describes the changes in the EAP-TLS-KS protocol in case an RSA signature key is used as the public roaming key and this key is used to sign the `Server-Key-Exchange` message which includes the ephemeral DH key part (Server-Pub-DH) generated by HN. Upon receipt of the `Client-Hello` message, HN computes the hash value

$$h(m) = h(\text{Client.RAND}||\text{Server.RND}||\text{Server-Pub-DH})$$

and half signs it to $h(m)^{-d_{HN_i}}$. HN then constructs the `EAP-Request` message including `Server-Hello`, `Server-Certificate` (with the RSA public encryption key as roaming key and FN's identity as attribute), `Server-Key-Exchange`, `Client-Certificate-Request`, and `Server-Done`. HN includes the half signed hash in `Server-Key-Exchange`.

Upon receipt of this message, FN completes the RSA signature on $h(m)$ by computing $s = h(m)^{-d_{HN_i}} \cdot h(m)^{2d_{FN_i}}$. It then replaces the half signed message with s in the `Server-Key-Exchange` message and forwards the information to MD. MD checks that the public-key in the roaming certificate is the one it has pre-installed. It generates its own public and secret ephemeral DH key values, computes the DH key from its own private DH value and the server's public DH value and generates the master key. MD then sends the respective EAP-TLS message to FN including its public DH value in the `Client-Key-Exchange` message.

HN verifies MD's certificate and MD's signature on `Client-Certificate-Verify`. Note that HN has to generate the complete signature on the hash value included in the `Server-Key-Exchange` message and use it to replace the half signed hash sent previously before it can compute the hash value of all messages sent and received so far. Otherwise, the signature will not be correct as MD received the complete signature (and not just the half signed message) included in the `Server-Key-Exchange` message from FN.

If HN can verify MD's signature, then HN sends the `EAP-TLS-KS-Success` message to FN. The receipt of this message assures FN of the correctness of MD's identity. FN now generates the DH key as well as the master key

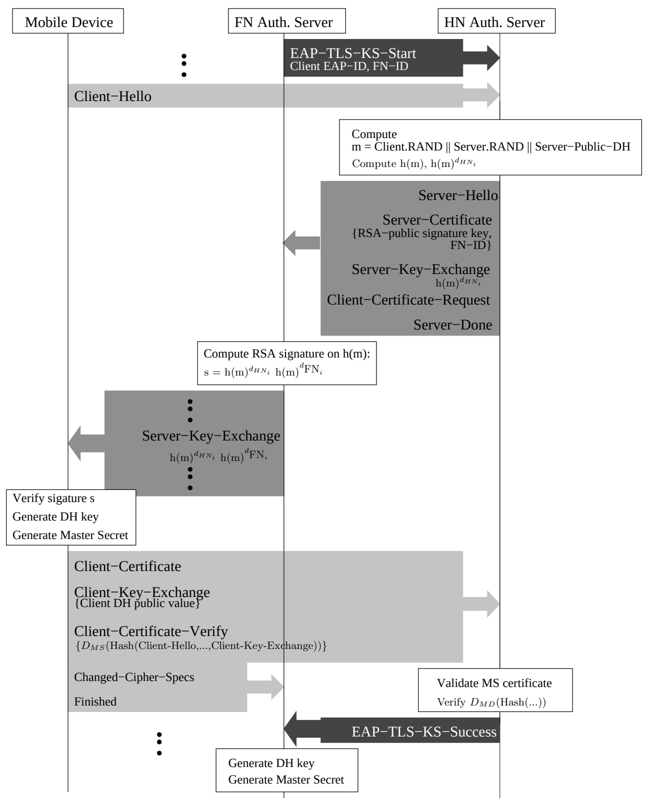

Figure 7: EAP-TLS-KS with DHE-RSA

and eventually completes the EAP-TLS-KS protocol as in the original EAP-TLS protocol. As in the RSA encryption-based case, MD is assured of the correctness of FN's identity by the correctness of the encryption of the `Finished` message received from FN.

4.3.3 DHE-DSS Case

Figure 8 describes the changes in the EAP-TLS-KS protocol in the case a DSS signature verification key is used as the public roaming key. HN and FN jointly sign the server's public DH key part using the distributed DSS signature described previously. The protocol is almost identical to the DHE-RSA case except that FN has to send the ephemeral DSS key part $\alpha^{\mathcal{K}_{FN_i}}$ to HN in the `EAP-TLS-KS-Start` message. In both the DHE-based protocols any FN_i can request HN to partly sign DH ephemeral keys regardless of any MD requesting to access FN_i. However, this can be noticed by HN if no `Client-Certificate-Verify` message follows the `Server-Key-Exchange` message. Moreover, as the hash value of the ephemeral DH key that is signed with the DSS or RSA key includes the Server.RAND as well as the Client.RAND, FN cannot use the recovered half signed DH ephemeral key to fool MD.

4.3.4 Cost

We analyze the new protocol in comparison to two other usage scenarios of EAP-TLS for inter-provider roaming. The first one is to use the authentication server in HN and having FN act like an access point in the regular EAP-TLS protocol. This scenario is equivalent to the one depicted in Figure 3 when replacing the client with MD, the server with the authentication server in HN and placing the authentica-

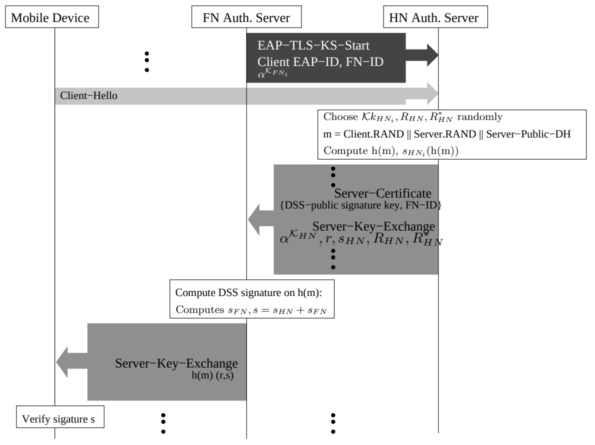
Mobile Device
FN Auth. Server
HN Auth. Server
EAP−TLS−KS−Start
Client EAP−ID, FN−ID
$\alpha^{\mathcal{K}_{FN_i}}$
Client−Hello
Choose $\mathcal{K}k_{HN_i}, R_{HN}, R^*_{HN}$ randomly
m = Client.RAND || Server.RAND || Server−Public−DH
Compute h(m), s_{HN_i}(h(m))
Server−Certificate
{DSS−public signature key, FN−ID}
Server−Key−Exchange
$\alpha^{\mathcal{K}_{HN}}, r, s_{HN}, R_{HN}, R^*_{HN}$
Compute DSS signature on h(m):
Computes $s_{FN}, s = s_{HN} + s_{FN}$
Server−Key−Exchange
h(m) (r,s)
Verify sigature s

Figure 8: EAP-TLS-KS with DHE-DSS

Table 1: Comparison

	EAP-TLS (FN)		EAP-TLS (HN)			EAP-TLS-KS		
	FN	MD	HN	FN	MD	HN	FN	MD
EAP mes.	5	4	5	9	4	5	3	4
MD sig. ver.	1		1			1		
Cert. val.	1	1	1		1	1		1*
KS mes.						1	1	
Key trans.			1					
	RSA-Based Key Generation							
RSA dec.	1		1			1	1	
RSA enc.		1			1			1
	DHE-RSA-Based Key Generation							
RSA sig.	1		1			1	1	
RSA sig ver.		1			1			1
	DHE-DSS-Based Key Generation							
DSS sig.	1		1			1	1	
DSS sig. ver.		1			1			1

tion server of FN in the middle to forward all traffic between MD and HN's AS. In the following we refer to this protocol as EAP-TLS (HN). Note that this setting requires a secure channel between FN and HN in order to transfer the master key from HN to FN.

In the second scenario, the authentication is fully delegated to FN, i.e., full control is delegated from HN to FN. This is equivalent to Figure 3 when replacing the server with FN's authentication server. In the following we refer to this scenario as EAP-TLS (FN).

Table 1 compares the three protocols EAP-TLS (HN), EAP-TLS (FN) and EAP-TLS-KS in terms of the number of EAP messages sent, signatures generated and verified and the number of messages encrypted or decrypted by MD, HN or FN. The three key generation types are listed separately.

The number of EAP messages MD has to send as well as the number of public-key operations MD has to perform are the same for all three protocols. Compared to the EAP-TLS (FN) protocol, the new protocol has the advantage that MD can authenticate FN without having to check the validity of any chain of certificates and the revocation status of these certificates. Instead, MD can use the pre-installed public roaming certificate and simply check whether it matches the received server certificate. If the certificate is revoked due to compromise, HN must immediately notify MD. In case the certificate expired, MD trusts HN not to engage in the authentication. This addresses an important problem with the other schemes where MD is required to check the validity of FN's certificate itself or delegate this task and wait for the respective response.

In comparison to EAP-TLS (FN), the new protocol shifts part of the load from FN to HN. In the new protocol, FN has to forward three EAP messages between MD and HN and has to send one additional EAP-TLS-KS message to HN. The load of verifying MD's signature on the `Certificate-Verify` message and the load of validating MD's certificate is shifted to HN.

Compared to EAP-TLS (HN), the new protocol significantly reduces the number of messages FN has to forward between MD and HN. The new protocol requires the forwarding of only three EAP messages as opposed to the nine messages in EAP-TLS (HN). The new protocol, however, requires two additional EAP-TLS-KS messages to be exchanged between HN and FN as well as an additional public-key operation by FN. Unlike in EAP-TLS (HN), the new protocol does not require any secure channel in order to transfer the master key from HN to FN.

4.4 Properties of EAP-TLS-KS

Complete Control by HN: HN fully controls every access requested by any of its subscribed MDs to any foreign network. HN furthermore fully controls the revocation status of both the roaming agreement with each FN as well as that of any MD. It can thus ensure that no successful authentication can take place after revocation. This eliminates the trust HN has to put into FN in other protocols, namely the trust, that FN correctly checks the revocation status of MD's certificate before granting access.

Proof of FN's ID to MD and Authentication of MD to FN: As HN uses a different key share for every FN, MD gains an indirect proof of FN's identity upon successful termination of the EAP-TLS-KS protocol. This proof of FN's identity enables MD to configure its own roaming policy locally, e.g., by excluding certain networks or keeping preference lists. This furthermore allows for a simple integration of an accounting initialization into the authentication procedure. Details will be discussed in Section 5.

Since authentication of roaming MDs requires HN to validate the certificate status, FN is assured that upon successful completion of the protocol HN approves MD to use FN's services and that HN is willing to reimburse FN for providing service to MD.

Elimination of Secure Channel between HN and FN: As opposed to many other schemes (e.g., [33, 45]), the new authentication protocol does not require the existence of a secure channel between HN and FN. This is due to the key splitting which makes it unnecessary to transfer the master key from HN to FN.

Simple Integration of New Roaming Agreements: A new roaming agreement between HN and a new foreign network FN$_{l+1}$ requires the generation of a new pair of shares of the secret roaming key. In the RSA cases, HN simply generates two new shares of the secret roaming key d, namely $d_{HN_{l+1}}$ and $d_{FN_{l+1}}$, distributes $d_{FN_{l+1}}$ to FN$_{l+1}$. In the DSS case, it generates a new pair of shares of the secret roaming key a, namely $(a_{HN_{l+1}}, a_{FN_{l+1}})$. It keeps both shares to itself and distributes $a_{FN_{l+1}}$ to FN$_{l+1}$. In both cases neither does the provider have to change the roaming key pair nor does it have to update or change any of the already distributed shares. Our scheme thus accommodates expansion to an arbitrary number of roaming agreements. The security of the scheme does not depend on the number of FNs the home provider has roaming agreements with. No adjustments are necessary for MD in order to allow for successful authentication to a new FN upon roaming.

Efficient Revocation of Agreements and Subscriptions: In order to revoke the roaming agreement with FN, HN simply marks the respective shares for that FN as revoked. Incoming authentication requests for the revoked FN are then no longer co-signed or half decrypted by HN. There is no need for HN to change the public roaming key.

The revocation status of the certificates for MDs is maintained by HN. Consequently, no FN is required to check the status of MD. Instead, revocation of MD's certificate is efficiently implemented by HN refusing to co-sign or half-decrypt authentication requests for revoked MDs.

Simple Handling of Compromised Keys: In case the key of a particular FN has been compromised, HN marks the corresponding shares as invalid. In particular, the compromise of the key share of an individual FN does not require the generating of a new roaming key pair. This is due to the fact that all FNs have individual shares.

In case the secret roaming key itself is compromised, HN has to immediately notify all its MDs of the revocation of the current roaming certificate and distribute a new one. However, it is not necessary for HN to provide the FNs with new shares. Consequently, the burden of expensive secret key share distribution in case of a compromised key is eliminated. In the following we discuss the details of handling compromised keys for the different roaming key types:

RSA Cases: The secret roaming key is a secret RSA key d and the public roaming key is a public RSA key pair (e, n). Let d^* be the new secret roaming key to be split with the FNs and let (e^*, n^*) be the corresponding public RSA key pair with $n^* \neq n$.[7] Then, HN determines $\delta = d - d^* \mod \varphi(n^*)$ as well as $\delta_i = d + w_i - d_{FN_i} \mod \varphi(n^*)$ and replaces its own old shares with the new shares

$$d^*_{HN_i} = 2(w_i - \delta - \delta_i) + d^* \mod \varphi(n^*)$$

for $i = 1, \ldots, l$. Consequently, any pair $(d^*_{HN_i}, d_{FN_i})$ can now be used to reconstruct the new roaming key d^* by:

$$\begin{aligned} -d^*_{HN_i} + 2d_{FN_i} &= -2(w_i - \delta - \delta_i) - d^* \\ &\quad + 2(w_i - \delta + d^* - \delta_i) \\ &= d^* \mod \varphi(n^*) \end{aligned}$$

In fact, the splitting of the new key d^* is done in the same way as the splitting of d by replacing w_i with $w_i - \delta - \delta_i$. That is, the random contribution of w_i is now provided by $\delta_i = d + w_i - d_{FN_i} \mod \varphi(n^*)$. If an attacker knows d and can thereby factor n, he can also learn w_i by collaborating with FN$_i$. However, the attacker cannot compute δ or δ_i as FN$_i$ does not know $\varphi(n^*)$. Even if two or more FNs collaborate, they cannot compute δ as each collaborating FN$_i$ contributes an unknown variable δ_i.

DSS Case: Splitting of a new secret roaming key $\alpha^{x^*} = a^*$ is obtained by determining a new additive splitting for x^*. That is, HN chooses a new $x^* \in \{1, \ldots, q-1\}$ and determines $\delta = x - x^* \mod q - 1$. It then computes

$$x^*_{HN_i} = x_{HN_i} + \delta \mod q - 1$$

and determines $a^* = \alpha^{x^*}$ as the new secret key and $a^*_{HN_i} = \alpha^{x^*_{HN_i}}$. Consequently, $a^*_{HN_i} \cdot a_{FN_i} = \alpha^{-x^*_{HN_i} + 2x_{FN_i}} = \alpha^{x - \delta} = \alpha^{x^*}$. Thus, HN and FN$_i$ now multiplicatively share the new secret key a^*.

It is important to note, that from a, a_{HN_i}, and a_{FN_i} the values x, x_{HN_i}, and x_{FN_i} cannot be recovered as long as the discrete logarithm assumption holds. Thus, if the key a is recovered by an attacker, x, x_{HN_i}, and x_{FN_I} are not affected. Consequently, the attacker can learn nothing about a^*.

An attacker who recovers a cannot use his knowledge on past signatures to compute new ones. As he knows nothing about the key a^*, he is in exactly the same situation as any signer with a secret key that tries to forge signatures of another signer using the same public parameters. If the attacker could use his knowledge on past signatures to compute valid signatures without knowledge of a^*, each signer could use his own past signatures to sign on behalf of someone else in DSS. The new splitting therefore is as secure against signature forgery as the original DSS signature scheme.

Simple Update of Keys: Updating of keys can be done in the same way as in the case of compromise.

5. ACCOUNTING

EAP-TLS-KS can be combined with any accounting method that WISP may decide to use, in particular those methods already in use. One of the shortcomings of currently used accounting methods is that they are restricted to per day or per session billing. In order to provide alternate subscription models, along with incontestable and more fine-grained billing, future systems will need to integrate more advanced accounting methods such as micropayment schemes. However, micropayment schemes designed to pay for streaming data services generally suffer from a relatively expensive initialization phase in order to achieve efficient payment [39]. This is different in the context of EAP-TLS-KS as the expenses of the initialization phase can be efficiently offset by means of the authentication protocol itself.

In the following we present a method to integrate the initialization phase of a slightly modified, more secure, version of the micropayment system introduced in [14] into EAP-TLS-KS. The integration of the micropayment scheme requires a single additional message in the EAP-TLS-KS protocol as the authentication and tariff commitment messages of the original protocol can elegantly be replaced with assurances from the EAP-TLS-KS protocol.

5.1 Buttyán and Hubaux's Protocol

The protocol presented in [14] is based around the idea of a customer care agency issuing a ticket to the user. The user then presents the ticket to a service provider to prove that the customer care agency will pay for the services provided.

The message flow of the accounting scheme is illustrated in Figure 9. The first stage in the accounting scheme is ticket acquisition. The user sends a ticket request including his identity U_{id} and a random number r_0 to the customer care agency. The agency A creates a ticket

$$T = (A_{id}, \underbrace{sn, c_n, publicDH_u, K_u}_{=:z}, K_a^{-1}(z))$$

with serial number sn, a value c_n, a public DH value for the

[7]It is important to ensure that n^* is different from n. Otherwise, since an attacker who knows d can factor n and thereby knows $\varphi(n)$ could also easily compute d^* by inverting e^* modulo $\varphi(n)$ in case $n^* = n$

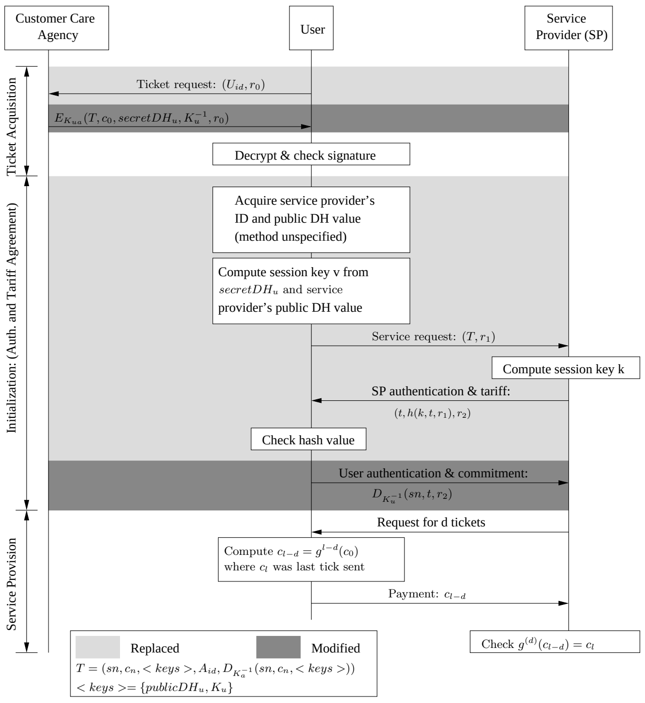
Customer Care Agency
User
Service Provider (SP)
Ticket Acquisition
Ticket request: (U_{id}, r_0)
$E_{K_{ua}}(T, c_0, secretDH_u, K_u^{-1}, r_0)$
Decrypt & check signature
Initialization: (Auth. and Tariff Agreement)
Acquire service provider's ID and public DH value (method unspecified)
Compute session key v from $secretDH_u$ and service provider's public DH value
Service request: (T, r_1)
Compute session key k
SP authentication & tariff:
$(t, h(k, t, r_1), r_2)$
Check hash value
User authentication & commitment:
$D_{K_u^{-1}}(sn, t, r_2)$
Service Provision
Request for d tickets
Compute $c_{l-d} = g^{l-d}(c_0)$ where c_l was last tick sent
Payment: c_{l-d}
Check $g^{(d)}(c_{l-d}) = c_l$
Replaced
Modified
$T = (sn, c_n, < keys >, A_{id}, D_{K_a^{-1}}(sn, c_n, < keys >))$
$< keys >= \{publicDH_u, K_u\}$

Figure 9: Buttyán and Hubaux Protocol

user and a public key K_u.[8] In addition, these parameters are signed with the agency's signature key K_a^{-1}. The agency then sends T together with the corresponding secret DH value $secretDH_u$, the secret key K_u^{-1}, and c_0 to the user. The message is encrypted under a key K_{ua} shared by the agency and the user.

The second stage is ticket usage. The user acquires the service provider's identity and public DH key exchange parameter in a way not further specified in [14]. It uses the provider's public DH value and its private one to compute a session key k. The user then sends a service request to the service provider containing T. The service provider can now compute k using the public DH value in T and its private DH value. The next message authenticates the service provider to the user and informs the user of the tariff t. The user's response authenticates the user and provides the user's commitment to the ticket and tariff which the service provider stores for use in billing.

Payment is based on a publicly known one-way hash function g and a chain of hash values starting with c_0 and ending with $c_n = g^{(n)}(c_0)$. At any point during service provisioning, the user has supplied the service provider with $n - l$ ticks by revealing c_l. If the service provider requests another b ticks the user sends $c_{l-b} = g^{(l-b)}(c_0)$ to the service provider. The service provider can then check whether $c_l \overset{?}{=} g^{(b)}(c_{l-b})$.

Billing takes place offline. The service provider simply presents the customer care agency with the ticket's serial number, the user's commitment, and the latest value of the hash chain it received from the user.

5.1.1 *Malicious Service Provider Attack*

Buttyán and Hubaux's scheme is vulnerable to the following attack. As anyone can observe a user's commitment to

[8]To maintain readability, we stick to the notation used in [14] at the cost of conflicts with the notation used in previous sections.

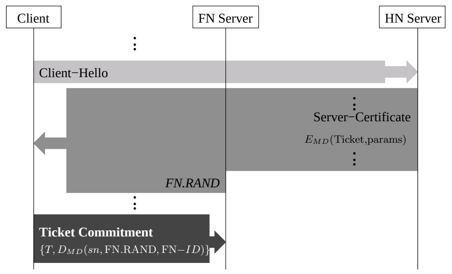

Figure 10: EAP-TLS-KS with Micropayment Initialization

the serial number sn of a ticket and the ticks c_l, an eavesdropper could present these to the customer care agency and request payment. In [14], the service provider signs its requests to be reimbursed before it sends them to the customer care agency in the clearance phase. This makes it impossible for anyone except authorized service providers, whose signature the customer care agency can verify, to get paid. However, a malicious service provider could collect ticket commitments and the corresponding ticks c_l which the user issues while using another service provider's service and then present these to the customer care agency. The malicious service provider could then be paid for a service it did not provide. Moreover, the customer care agency would claim that the user tried to use the same ticket on two different service providers and would accuse the user of fraud.

We suggest to protect against this attack by adding the service provider's identity into the user's commitment such that only this service provider can present the commitment to the customer care agency [1].

5.2 Modifications for Use with EAP-TLS-KS

The entities in the micropayment scheme are mapped to the following entities in the roaming scenario: the user is mapped to the mobile device, the role of the customer care agency is provided by the home network, and the service provider is the foreign network. Accordingly, Figure 10 describes the modifications necessary to the EAP-TLS-KS protocol in order to include micropayment initialization. Since HN receives the identity of MD in the `EAP-Request-Identity` message and the random number Client.RAND in the `Client-Hello` message, there is no need for an explicit ticket request in the modified protocol. As EAP-TLS-KS provides mutual authentication, one can remove the two public keys from the ticket in the original micropayment scheme. However, in order to prevent an eavesdropper from replaying a ticket to a different FN, the ticket must include the client's identity MD-ID. Thus, the complete structure of the ticket is:

$$T = (sn, c_n, \text{MD-ID}, D_{K_{HN}^{-1}}(sn, c_n, \text{MD-ID}))$$

where sn is the ticket's serial number, c_n is the final value of the one-way hash chain and K_{HN}^{-1} is the corresponding private part of the server-certificate sent from HN to MD. This signature allows MD to verify the origin of the ticket.

The two secret keys corresponding to the eliminated public keys can be removed from HN's response. Consequently, HN sends $E_{K_{MD}}(T, c_0, \text{Client.RAND})$ to MD which is encrypted with the public key K_{MD} corresponding to a decryption key held by the MD.[9] The corresponding message is appended to the Server-Certificate message in EAP-TLS-KS. As this message is forwarded through FN's server, FN appends a newly generated random number FN.RAND.[10] FN.RAND corresponds to the random number r_2 in the original protocol, and its inclusion in the commitment by the MD serves the same purpose; to guarantee the freshness of the commitment to FN.

The original protocol works under the assumption that the tariff could be different each time the user connects to a service provider, and that the user would have to agree to this tariff in its commitment. We assume that the tariff will be negotiated by the WISPs ahead of time as part of their roaming agreements. Therefore, we can eliminate the part of the protocol used to inform MD of the tariff. Also, the authentication during this part of the protocol is replaced by the authentication in EAP-TLS-KS. Once the `EAP-Success` messages are exchanged, i.e., MD and FN have authenticated each other, MD commits to the use of ticket T as payment to FN. It does so by sending FN the ticket and its signature of the serial number, the random number FN.RAND as well as the FN's identity:

$$\{T, D_{MD}(sn, \text{FN.RAND}, \text{FN-ID})\}$$

This commitment is the only additional message the initialization of the accounting scheme requires if integrated in EAP-TLS-KS. In a final step, FN verifies the signature and stores the message for the clearance process.

Service provisioning, clearance and billing works as in the original scheme, except that HN checks FN's signature against the identity included in the ticket commitment.

Security Analysis: Fabricating or modifying tickets by either an outside attacker or a valid MD is prevented by HN signing the data in the ticket. Similarly, MD's signature on the commitment along with the signed client identity in the ticket prevent an eavesdropper from using an intercepted ticket.

Reuse of a previously used ticket by MD will be detected (at the latest) in the clearance and billing stage. HN can prove that MD cheated from the existence of two commitments with the same serial number but differing FN.RAND's. If MD attempts to use the same ticket more than once with the same FN, then FN will be able to detect the fraud since it is required to retain copies of previous commitments for billing purposes. If MD attempts to use the same ticket with two different FNs, then this fraud is not detected until billing takes place. However, since MD is guaranteed to be caught, it is a matter of imposing a large enough penalty to offset any potential gain and thus discourage MD from reusing tickets.

By including FN's identity in MD's ticket commitment, no one but FN can redeem the ticket commitment in combination with any tick by presenting the information to HN.

Should FN decide to suddenly stop providing service, it will gain very little as MD will not release any more ticks. Due to the use of a one-way hash function it is impossible for FN to forge ticks. Likewise, if the user stops sending ticks, FN may simply stop providing service. Consequently, it may lose at most one pay period (or whatever tolerance is set as acceptable).

[9]It is important to note that in the original EAP-TLS-KS protocol, MD is not required to hold a decryption key. Therefore, MD must be issued a decryption key in order to allow for the use of the micropayment extension.

[10]FN.RAND must be excluded from the signed message hash sent by MD to HN since HN has no knowledge of it (nor does it need any).

6. RELATED WORK

6.1 Inter-Provider Roaming in Public WLANs

Overviews: In [9] Balachandran et al. discuss open questions and challenges related to WLAN hotspot providers with an emphasis on roaming issues and security. In [46] Wang et al. discuss and analyze the security mechanisms UAM, 802.1X, PANA and USIM-based authentication for wireless hotspot providers and inter-provider roaming.

Web-Based Authentication Methods: The most widely-used authentication protocol by WISPs is the web-based universal access method UAM. This method has been shown to have several vulnerabilities [46]. A renegade access point connected to a web server with a valid SSL certificate can be set up in the hotspot and trick users into divulging their authentication credentials. Furthermore, UAM is vulnerable to dictionary attacks. Also, a malicious MD can spoof the address pair (MAC/IP address) of an already authenticated MD to conduct service theft. Another web-based authentication method is CHOICE [8] which is secure against address spoofing. However, it uses the MS-Passport technology [36], which makes it platform dependent. A proprietary security sublayer between the link layer and the IP layer further restricts the application area of CHOICE. In contrast, the protocol proposed in this paper is based on publicly-available technologies only. Spinach [6] offers a web-based interface to a Kerberos authentication service and aims to not only protect public wireless access points but also to secure public network ports. It is designed to be flexible with respect to the use of other authentication methods if desired. Unfortunately it is vulnerable to address spoofing.

Other Proposals: Salgarelli et al. [45] suggest an authentication protocol EAP-W-SKE that minimizes the number of round-trip message exchanges between FN and HN to one round-trip. However, this requires a secure channel between HN and FN for key transfer. In [33] Matsunaga et al. propose a single sign-on authentication architecture that is based on 802.1X and EAP-TLS for PKI-based network authentication. It can be combined with any web-based authentication method for MD authentication. Due to the use of 802.1X, their architecture is secure against address spoofing. Yet, the web-based user authentication mechanisms require a secure channel between the local web server and the user's identity server of choice. Our protocol does not require the existence of any secure channel between any of the components of the visited FN and HN. Moreover, in [33] it is assumed that MDs can check the validity of any public-key certificate presented by FN as part of the EAP-TLS protocol. The problems that arise from certificate chain discovery and validation by MD are not addressed in [33].

Some protocols have been suggested for inter-provider roaming that use public-key-based methods to authenticate both the network and MD. Gu et al. made an explicit suggestion for WLAN [21] whereas Bayarou et al., for example, suggest a framework for wireless networks in general [10]. The authentication protocol in [21] shares the aforementioned deficiencies concerning costly discovery, verification and validation of certificate chains by an offline MD. In [10] these shortcomings are addressed by delegating certificate chain discovery and validation to a trusted server. This solves the offline problem, but causes additional round-trips to the trusted server.

6.2 Distributed Signatures

Secret Sharing Scheme: For the construction of a secret sharing scheme that represents our non-threshold composite access structure we make use of an approach presented and proved in [32]. The authors show that a composite access structure $\Gamma_0[(t_1, n_1), (t_2, n_2), ..., (t_l, n_l)]$ allows for a vector space construction if the initial access structure Γ_0 itself allows a vector space construction. The proof is constructive and we use it in a straight-forward manner to construct our linear secret sharing scheme.

Geer and Yung suggest alternative applications of threshold cryptography in [19]. Although this work does not address inter-provider roaming, this paper inspired our work.

Distributed RSA: Distributed RSA decryption and signatures were first suggested and analyzed in [13] and [17]. We use these methods in EAP-TLS-KS in a straight-forward manner. In [12], Boneh et al. use a semi-trusted mediator in conjunction with two-party RSA signature schemes and cryptosystems to facilitate user certificate revocation. In this paper, we refer to their security analysis for the distributed RSA decryption. In [30], MacKenzie et al. use distributed RSA signatures to secure PIN-protected or password-protected private keys against off-line dictionary attacks in order to achieve capture resilience. We refer to their formal security analysis for the distributed RSA signature scheme.

Distributed DSS: Distributed DSS signatures are particularly hard to construct since not only a long term secret key, but also the ephemeral key has to be split between signers. A fully symmetrical two-party DSS signature generation scheme was presented by MacKenzie et al. in [31]. This scheme requires a semantically secure public-key encryption scheme to be implemented between the two signers that exhibits a specific homomorphic property as, e.g., in the cryptosystems of Pallier [38] or Okamoto et al. [37]. The purpose of the encryption scheme is, to allow one party to reveal his share encrypted with its public key to the other party. The other party can use the encrypted share to generate an encrypted signature and then send this back to the first party, who finally decrypts the full signature. The encryption scheme thus enables the symmetry of the two-party signature. We assume an asymmetric setting in which HN keeps the complete key as well as FN's shares for several reasons. First, it allows HN to engage in new roaming agreements. Second, it enables simple key update and key compromise handling. Finally, it allows HN to use the full secret key if MD "roams" to HN. The use of MacKenzie et al.'s distributed DSS version in our setting would thus generate unnecessary overhead. In contrast, the distributed DSS signature scheme introduced in this paper is tailored to the specific setting of roaming in which one of the signing parties has complete power over the other. In the first work on distributed DSS signatures [28], Langford presented a $(2, l)$ threshold DSS signature for $l \geq 3$. This construction was generalized by Gennaro et al. to a (t, n) threshold signature with $n \geq 2t + 1$ in [20]. These schemes are not directly applicable in our scenario as we require a $(2, 2)$ signature scheme. However, our distributed version is similar to the $(2, 3)$ threshold DSS signature presented in [28]. Langford blinds the shares of each party with random numbers and uses a third party to unblind and finally compose partly signed messages into fully signed ones. We use a similar blinding with random numbers to conceal HN's share from FN.

6.3 Micropayment Schemes

Micropayment schemes are payment schemes designed to enable payments of small amounts. Application areas of such schemes include payments for web content like single visits to web pages and payments for streaming data such as music or video streams on a per minute basis. Overviews of micropayment schemes and electronic payment schemes in general can be found in [29] and [7].

Fine-grained billing for inter-provider roaming is one of the application areas for a special type of micropayment schemes, that are designed for subsequent small payments to the same vendor.

Schemes optimized for this purpose use subsequent values from a one-way hash chain rather than digital signatures on payments, in order to reduce the load imposed on the "bank" verifying subsequent payments. Using one-way hash chains in micropayment schemes for efficiency reasons was independently suggested by Anderson et al.[4], Pedersen [39], and Rivest and Shamir [43]. In [43] Rivest and Shamir present two systems PayWord and MicroMint. MicroMint eliminates public-key operations at the risk of forgery. This restricts the use of MicroMint to uncorrelated small payments. PayWord, [43] uses a hash chain where the root is digitally signed by the user. It is a credit system and as such has the disadvantage that a client can cause damage to the bank by purchasing in excess [3]. Anderson et al.'s NetCard [4] bypasses this problem by using the bank's signature on the root of each chain the user generates, thus requiring the bank to be online at the first use of each chain. Buttyán et al. use a similar setting, yet shift the chain generation as well as the signing of the root of the hash chain to the bank. This allows for the bank to be off-line. Additionally, Buttyán et al. integrate authentication and tariff negotiation directly into their accounting scheme.

All of the previously mentioned schemes introduce efficient service provisioning at the cost of expensive initialization. Horn and Preneel first demonstrate how a such a micropayment scheme could be integrated into an authentication protocol [23] to minimize the expense of the initialization phase. We use this idea to efficiently integrate Buttyán et al.'s scheme in EAP-TLS-KS. Their scheme is particularly suitable for our application for two reasons: First, the authentication assurances in the original protocol can easily be extracted and replaced with the EAP-TLS-KS assurances without affecting the security the actual payment system. Second, the off-line ticket acquisition of the original protocol can be integrated into our protocol without causing any additional messages.

Another approach to bypass the over-spending problem without requiring the bank to be online is to randomize the audit time point [18, 27]. In EAP-TLS-KS the home network has to be online during authentication. Consequently, a randomization of the audit would not provide any advantage in EAP-TLS-KS.

Another trend in the development of micropayment schemes is to randomize the payments in order to reduce the load on the bank and the vendor in case of single small payments. This idea goes back to [41] and was further developed in [29]. A recent suggestion in this direction is the Peppercoin micropayment scheme of [35] as well as its variations [42] and application [26]. In our scenario a user will typically use a WISP for longer than a few seconds. Thus we do not need a protocol that is optimized for only a single small payment.

7. OUTLOOK

Our protocol does not yet explicitly support anonymous roaming nor does it support quality-of-service dependent payment. Future work will explore how current research on these topics such as the ideas in [44] can be integrated into our protocol. Generalizing the key splitting approach to support roaming mediators in addition to pairwise roaming agreements is another direction for future work.

8. REFERENCES

[1] Personal Communication with L. Buttyán, May 2005.

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

[3] N. Adachi, S. Aoki, and Y. Komano. The security problems of Rivest and Shamir's PayWord scheme. In *Proceedings of IEEE CEC'03*, 2003.

[4] R. J. Anderson, C. Manifavas, and C. Sutherland. NetCard - A practical electronic-cash system. In *Security Protocols*, volume 1180 of *LNCS*, 1997.

[5] B. Anton, B. Bullock, and J. Short. Best current practice for wireless internet service provider (WISP) roaming. Wi-Fi Alliance - Wireless ISP Roaming (WISPr), February 2003.

[6] G. Appenzeller, M. Roussopoulus, and M. Baker. User-friendly access control for public network ports. In *Proceedings of IEEE INFOCOM'99*, 1999.

[7] N. Asokan, P. Janson, M. Steiner, and M. Waidner. State of the art in electronic payment systems. *IEEE Computers*, (30), September 1999.

[8] P. Bahl, A. Balachandran, and S. Venkatachary. Secure wireless internet access in public places. In *Proceedings of IEEE ICC'01*, 2001.

[9] A. Balachandran and G. M. Voelker. Wireless hotspots: Current challenges and future directions. In *Proceedings of ACM WMASH'03*, 2003.

[10] 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, June 2004.

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

[12] D. Boneh, X. Ding, and G. Tsudik. Fine-grained control of security capabilities. *ACM Transactions of Internet Technology*, 4(1), February 2003.

[13] C. Boyd. Digitial multisignatures. In *Conference on Cryptography and Coding*, 1986.

[14] L. Buttyan and J.-P. Hubaux. Accountable anonymous access to services in mobile communication systems. In *Symposium on Reliable Distributed Systems*, 1999.

[15] T. Clancy and W. Arbaugh. EAP Password Authenticated Exchange (EAP-PAX). Internet Society draft-clancy-eap-pax-03, April 2005.

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

[17] Y. Frankel. A practical protocol for large group oriented networks. In *Advances in Cryptology - EUROCRYPT'89*, LNCS, 1989.

[18] E. Gabber and A. Silberschatz. Agora: A minimal distributed protocol for electronic commerce. In *Proceedings of the USENIX Workshop on Electronic Commerce*, 1996.

[19] D. Geer and M. Yung. Split-and-delegate: Threshold cryptography for the masses. In *Proceedings of FC'02*, volume 2357 of *LNCS*, 2002.

[20] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *Advances in Cryptology - EUROCRYPT'96*, volume 1070 of *LNCS*, 1996.

[21] 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 ACISP'03*, volume 2727 of *LNCS*, 2003.

[22] H. Haverinen and J. Salowey. Extensible Authentication Protocol method for GSM Subscriber Identity Modules (EAP-SIM). Internet Society, draft-haverinen-pppext-eap-sim-16.txt, December 2004.

[23] G. Horn and B. Preneel. Authentication and payment in future mobile systems. In *Proceedings of ESORICS'98*, volume 1485 of *LNCS*, 1998.

[24] IEEE. IEEE 802.1X - Port-based network access control, June 2001.

[25] IEEE. IEEE 802.11i - Specification for enhanced security, July 2004.

[26] M. Jakobsson, J.-P. Hubaux, and L. Buttyán. A micro-payment scheme encouraging collaboration in multi-hop cellular networks. In *Proceedings of FC'03*, volume 2742 of *LNCS*, 2003.

[27] S. Jarecki and A. Odlyzko. An efficient micropayment system based on probabilistic polling. In *Proceedings of FC'97*, volume 1318 of *LNCS*, 1997.

[28] S. K. Langford. Threshold DSS signatures without a trusted party. In *Advances in Cryptology - CRYPTO'95*, volume 963 of *LNCS*, 1995.

[29] R. J. Lipton and R. Ostrovsky. Micropayments via efficient coin-flipping. In *Proceedings of FC'98*, volume 1465 of *LNCS*, 1998.

[30] P. MacKenzie and M. K. Reiter. Networked cryptographic devices resilient to capture. In *Proceedings of IEEE Symposium on Security and Privacy*, 2001.

[31] P. MacKenzie and M. K. Reiter. Two-party generation of DSA signatures. In *Advances in Cryptology - CRYPTO'01*, volume 2139 of *LNCS*, 2001.

[32] E. Martinez-Moro, J. Mozo-Fernandez, and C. Munuera. Compounding secret sharing schemes. *Australian Journal of Combinatorics*, 30, September 2004.

[33] Y. Matsunaga, A. S. Merino, T. Suzuki, and R. H. Katz. Secure authentication system for public WLAN roaming. In *Proceedings of ACM WMASH'03*, September 2003.

[34] A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[35] S. Micali and R. Rivest. Micropayments revisited. In *Proceedings of CT-RSA'02*, 2002.

[36] Microsoft passport network. `http://www.passport.com`.

[37] T. Okamoto and S. Uchiyama. A new public-key cryptosystem, as secure as factoring. In *Advances in Cryptology - EUROCRYPT'98*, volume 1403 of *LNCS*, 1998.

[38] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology, EUROCRYPT'99*, volume 1592 of *LNCS*, 1999.

[39] T. P. Pedersen. Electronic payments of small amounts. In *Security Protocols*, volume 1180 of *LNCS*, 1996.

[40] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Authentication Dial in User Services (RADIUS). RFC 2865, June 2000.

[41] R. L. Rivest. Electronic lottery tickets as micropayments. In *Proceedings of FC'97*, volume 1318 of *LNCS*, 1997.

[42] R. L. Rivest. Peppercoin micropayments. In *Proceedings of FC'04*, volume 3110 of *LNCS*, 2004.

[43] R. L. Rivest and A. Shamir. PayWord and MicroMint: Two simple micropayment schemes. In *Security Protocols*, volume 1180 of *LNCS*, 1996.

[44] N. B. Salem, J. P. Hubaux, and M. Jakobsson. Reputation-based Wi-Fi deployment protocols and security analysis. In *Proceedings of ACM WMASH'04*, 2004.

[45] L. Salgarelli, M. Buddhikot, J. Garay, S. Patel, and S. Miller. Efficient authentication and key distribution in wireless IP networks. *IEEE Wireless Communications Magazine*, 2003.

[46] H. Wang, R. Prasad, A. P. Schoo, M. Bayarou, K. and S. Rohr. Security mechanisms and security analysis: Hotspot WLANs and inter-operator roaming. In *Proceedings of ACM WMASH'04*, 2004.