Man-in-the-Middle in Tunnelled Authentication Protocols

Extended Abstract*

N. Asokan, Valtteri Niemi, and Kaisa Nyberg

Nokia Research Center, Finland
{n.asokan,valtteri.niemi,kaisa.nyberg}@nokia.com

Abstract. Deploying a new security protocol is expensive. This encourages system designers to look for ways of re-using existing infrastructure. When security protocols and components are re-used, it is critical to re-examine the security of the resulting system as a whole. For example, it has become a standard paradigm to run a legacy client authentication protocol within a secure tunnel. The commonest example of such composition is the use of HTTP authentication inside a TLS tunnel.
In this paper, we describe a man-in-the-middle attack on such protocol composition. The vulnerability arises if the legacy client authentication protocol is used both in tunnelled and untunnelled forms. Even when the client authentication protocol and the tunnel protocol are both secure, composing them in the customary manner results in an insecure system.
We propose a solution to this problem by using a cryptographic binding between the client authentication protocol and the tunnel protocol.

1 Introduction

When new services and applications are developed, their security requirements and trust assumptions may necessitate designing and deploying new security protocols. However, deploying a new security protocol is expensive. This encourages system designers to look for ways of re-using existing infrastructure. One form of reuse is to build on existing protocol components or frameworks so that the need to deploy new software is reduced. But the most difficult aspect of deployment is the cost of provisioning initial security associations, especially to end user devices. Consequently, there is considerable pressure to reuse security protocols and security context databases beyond their originally intended purposes and environments.
Re-use is generally considered good practice. However, when security protocols and components are reused, it is critical to re-examine the security of the resulting system as a whole. This is particularly important when different components of the composition have different endpoints, either in different layers in the same entity, or different entities altogether. The fact that there are no easy-to-use tools or methodologies to verify the correctness of security protocols makes protocol re-use a risky task.

* An earlier, longer version of this work appeared as a research report [5]

Faced with this difficulty, designers who needed to re-use legacy client authentication protocols and/or security associations have been customarily using an obvious approach to secure legacy client authentication protocols when there is a need to re-use them in a new scennario. They define an authentication protocol as a combination of two protocols: The legacy client authentication mechanism is run inside a secure tunnel. The most common example of this type of construction is the combination of a server-authenticated TLS tunnel [9] and the HTTP Digest Authentication mechanism [11]. Shortcomings of the legacy client authentication protocol in the new environment can apparently be offset by the protective tunnelling. For example, HTTP Digest Authentication may be based on a simple user-name/password; but the TLS tunnel will protect the password from any eavesdroppers. At the same time, the legacy authentication method may continue to be used in legacy environments. Therefore, at first glance, secure tunnelling appears to be a reasonable approach. This is why, as described in Section 2, this type of protocol composition has been widely used when there is a perceived need for reuse of legacy authentication mechanisms.

Unfortunately, when such a legacy client authentication protocol needs to be run both in the legacy environment as well as in a tunnelled environment, a man-in-the-middle attack becomes possible. Even when the client authentication protocol and the tunnel protocol are both secure, the composing them in the customary manner described above, results in an insecure system. In this paper, we describe the attack, and discuss how it can be avoided by cryptographically binding the inner and outer protocols. If the inner client authentication protocol produces a suitable key, this binding *does not require any changes* to the inner protocol. This is a very important requirement because the whole point of using legacy protocols is that they are already widely deployed.

In section 2 some background information is provided by highlighting the common tunnelling approach and using PEAP as an example to describe how this is instantiated in practice. Then the man-in-the-middle attack is described in section 3. Ways of removing the vulnerability are presented in section 4. The use of weak authentication methods is discussed in section 5. The implications of this attack are discussed in 6. The conclusions are summarized in section 7. Finally, a brief status update is provided in section 8.

2 IETF Drafts on Tunnelled Authentication Protocols

2.1 The general model

In this section, we present a general description of how tunnelled authentication protocols are usually constructed. Although we discuss examples that use extensible authentication protocol (EAP) as the inner client authentication protocol, the discussion is applicable for any tunnelled authentication protocol.

EAP, described in RFC2284 [17], is a standard framework for client authentication protocols. By using EAP in a system, it is possible to enable the system to support a number of legacy authentication schemes, including smart cards, Kerberos, public key mechanisms, One Time Passwords, cellular authentication mechanisms like GSM [14] or UMTS AKA (Universal Mobile Telecommunication System, Authentication and Key Agreement protocol) [3], and many others.

EAP is run between a client and a server. We call this server a *backend server*. There may be a a *front-end authenticator* between the client and the backend

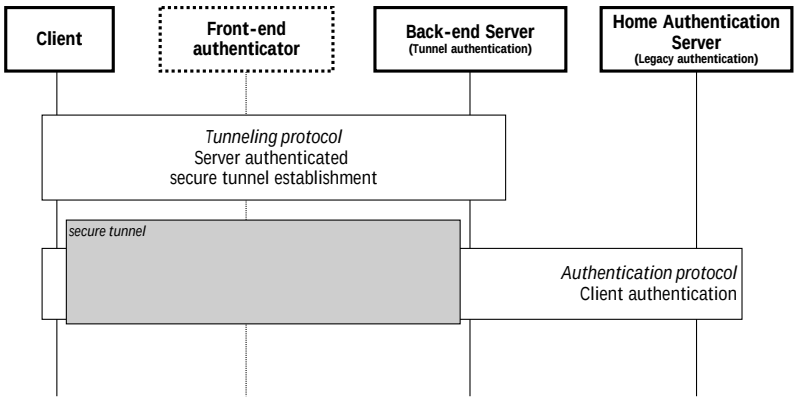

Fig. 1. Customary construction of a tunnelled authentication protocol

server. The front-end authenticator will simply forward authentication messages to the backend server. The backend server may use yet another server to help authenticate the client. We call this third server the *home authentication server*. Use of EAP allows new authentication methods to be developed without requiring deployment of new code on the front-end authenticator. The front-end authenticator acts as a "pass-through", and need not understand specific EAP methods.

Recently new protocols have been proposed in the Internet Engineering Task Force (IETF) for running EAP inside a server-authenticated tunnel. Examples of such protocols are PIC [23], PEAP [15], EAP-TTLS [12], POTLS [20], and most recently SLA [13], which led to the mechanisms for supporting legacy authentication methods in IKEv2 [10]. Some of these protocols, like PEAP, are motivated by a wish to correct perceived weaknesses of EAP, such as the lack of user identity protection and lack of a standardized mechanism for key exchange [15]. Others like PIC, POTLS and SLA are motivated by a desire to re-use legacy authentication credentials and databases for new applications. All of these new protocols are constructed in the same basic manner. First, a secure tunnel is set up using a suitable protocol like TLS. In most cases, the tunnel is also server-authenticated. Then the client authentication protocol is run inside this tunnel. This general model of this customary approach is illustrated in Figure 1. Other forms of tunnelled authentication protocols, such as HTTP Digest authentication inside a TLS tunnel, also conform to this general model.

2.2 Protected EAP

Protected EAP (PEAP) [15] is an example of the type of composition described in Figure 1: it wraps the EAP protocol messages within TLS [9]. It claims to provide user anonymity and built-in support for key exchange.
The relationship between the EAP peer (client), front-end authenticator, known as the "network access server" (NAS) in PEAP, and a backend server, known as the "back-end authentication server" in PEAP, is depicted in Figure 2. As described in the figure, the EAP conversation "passes through" the NAS on its way between the client and the back-end authentication server. While the authentication conversation is between the EAP client and the back-end authentication server, the NAS and back-end authentication server need to have established trust for the conversation to proceed.

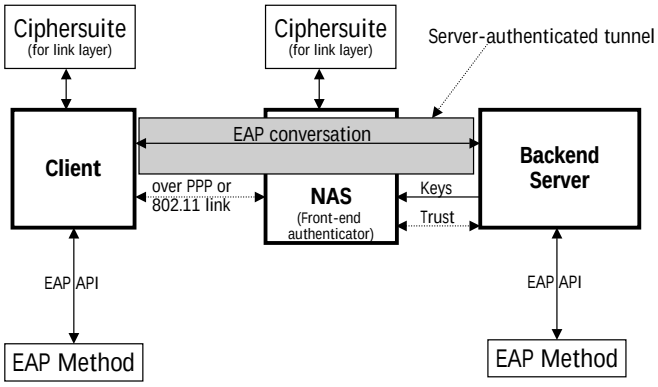

Fig. 2. Relationship between EAP client, back-end authentication server and NAS in PEAP [15]

The client and the back-end server first set up a TLS channel over EAP. The client authentication protocol messages between the client and the back-end server are encrypted and integrity protected within this TLS channel. The NAS does not have knowledge of the TLS master secret derived between the client and the back-end authentication server, and cannot decrypt the PEAP conversation. The back-end server derives master session keys from the TLS master secret via a one-way function and conveys them to the NAS which can then use these session keys to protect subsequent link-layer communication between it and the client.

The session key is transported from the server to the NAS using e.g. Radius [22] or DIAMETER [8] attributes.
In a recent Internet draft, EAP SIM GMM authentication [7], an application of PEAP to GSM authentication was presented. The same approach can be used to combine PEAP with AKA which is the client authentication mechanism in the third generation UMTS networks. The message flow in PEAP with EAP AKA client authentication in the context of wireless local area network (WLAN) access authentication is depicted in Figure 3.
The WLAN authentication server is in the role of the back-end server. The terminal has a smart-card containing a secret key. The same key is available in the subscriber database of HSS (Home Subscriber Server) located in the home network of the subscriber. The terminal sends a cellular identity like the International Mobile Subscriber Identity (IMSI) as part of EAP AKA. The WLAN server uses a protocol like DIAMETER to send the IMSI to the HSS. HSS returns an AKA authentication vector, which is a quintuplet containing a challenge (RAND), an authenticator for the challenge (AUTN), expected response (XRES), and session keys IK (for integrity) and CK (for confidentiality) to the WLAN server. The WLAN server forwards RAND and AUTN to the terminal via EAP AKA. The terminal can now verify AUTN to confirm that RAND comes from HSS, and compute its own response RES and the keys IK and CK. The terminal sends RES back to the WLAN server using EAP AKA. If RES and XRES are the same, WLAN server considers the terminal to be authenticated. The AKA protocol was defined as part of the third generation (3G) cellular standardization activities. More detailed descriptions of 3G protocols are available elsewhere, e.g., [16].
Again, only the TLS master secret is used to derive the session keys to be used to protect the WLAN link. The secret key material carried within the UMTS AKA [16] authentication quintuplets is not used.

3 Man-in-the-Middle Attack

Why can the tunnelling approach go wrong? There are two reasons:

- The legacy client authentication protocol is used in other environments, e.g., plain EAP without any tunnelling, or without any EAP encapsulation at all, e.g., direct use of one-time passwords, or cellular authentication protocols.
- The client cannot or does not properly authenticate the server (even when the authentication protocol is used *within* a supposedly server-authenticated tunnel).

The active attack by a Man-in-the-Middle (MitM) proceeds as follows:

1. MitM either waits for a legitimate device to start an untunnelled legacy remote authentication protocol or actively fools the legitimate device into initiating an untunneled remote authentication protocol. It then captures the initial message sent by the legitimate client.
2. MitM initiates a tunnelled authentication protocol with a backend server.
3. After the tunnel is set up between MitM and the backup server, the MitM starts forwarding legitimate client's authentication protocol messages through the tunnel.
4. MitM unwraps the legacy authentication protocol messages received through the tunnel from the backend server and forwards them to the legitimate client.

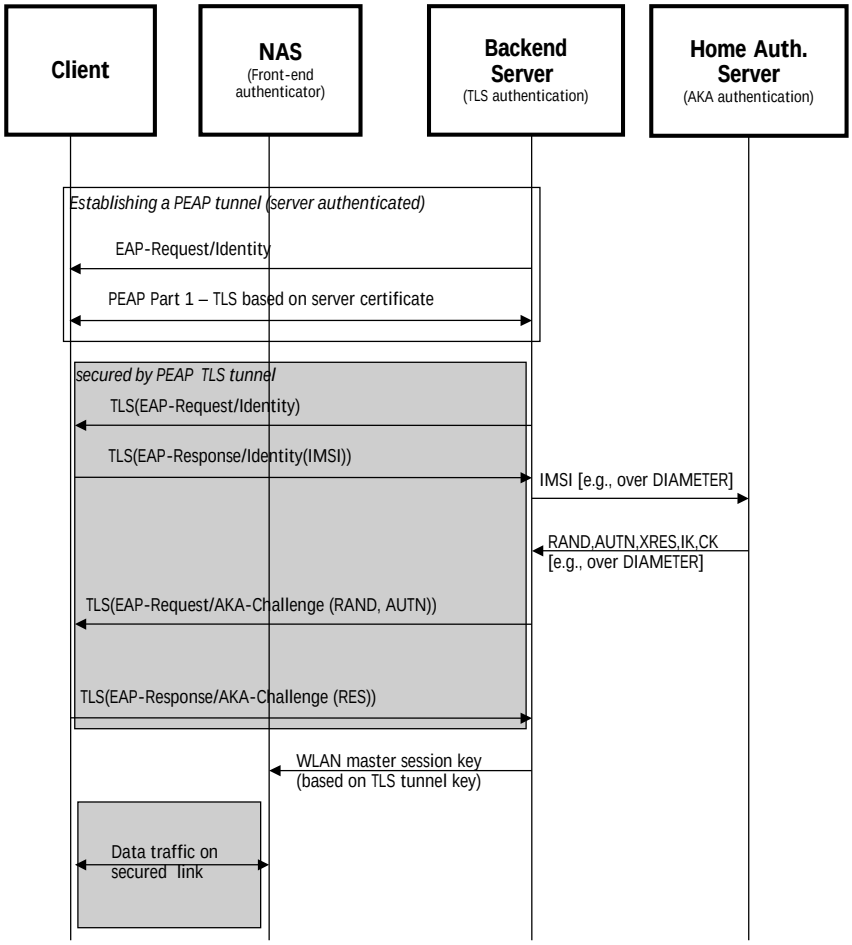

Fig. 3. PEAP with EAP AKA: Example Message Flow

5. After the remote authentication ended successfully, MitM derives the session keys from the same keys it is using for the tunnel.

Some legacy authentication protocols do not support mutual authentication or session key agreement. When they are used in an open environment, they are obviously vulnerable to MitM attacks. However, tunneling a generic authentication protocol framework like EAP makes the MitM attack possble even when the actual authentication method used within the framework does support mutual authentication and session protection. This situation arises because the backend server cannot be certain that the client in the legacy authentication protocol and the client endpoint of the tunnel are the same entity.

Figure 4 shows how the MitM attack works in PEAP with EAP AKA as the example client authentication protocol in a WLAN access authentication setting. The victim terminal assumes that the MitM is a UMTS radio access network. Note that the UMTS AKA protocol itself [16] and its EAP encapsulation [3] provide for mutual authentication of the user terminal and the radio access network: AUTN parameter is used to authenticate the source of the RAND challenge. The attempt to tunnel EAP AKA through PEAP effectively defeats this protection. In other words, even though the inner authentication protocol (EAP AKA) and the outer protocol (TLS) are both secure as such, composing them in the customary manner results in an insecure system.

Surprisingly, this type of protocol composition is very popular. As mentioned before, the most widespread of these may be the use of HTTP authentication [11] through a TLS tunnel (i.e., to an `https` URL). Many web sites use this type of a scheme for controlling access to their resources; for example, on-line access to bank accounts are usually controlled this way. Naturally, if the authentication method is usable without a TLS tunnel (e.g., to a plain `http` URL), a MitM attack is possible.

4 Avoiding the attack: Cryptographic Binding

The vulnerability discussed above is due to the fact that the legitimate client and the backend server had no way to verify that their peer in the client authentication protocol is the same as the entity at the other end of the outer tunnel which possesses the session keys at the end of the protocol. We can enable this verification either implicitly or explicitly. The goal of *implicit authentication* is to ensure that the only the legitimate parties are capable of gaining access to the correct session. In *explicit authentication* a separate authentication step is performed to verify that the entities possessing the master secrets from which the session keys are derived, are indeed the legitimate parties. In both cases, the authentication is provided by a cryptographic binding between the inner and the outer protocols.

These solutions are applicable to mutual authentication protocols that are constructed as a combination of two authentication protocols. The outer protocol is assumed to support the construction of a secure tunnel based on server authentication. As we saw, this is the case with all the protocols cited in section 2.

A pre-requisite to cryptographic binding is that the inner authentication protocol must produce a key S in one of two ways:

1. S is a session key resulting from a run of the inner protocol; i.e., it is an authentication *and* key agreement protocol; or

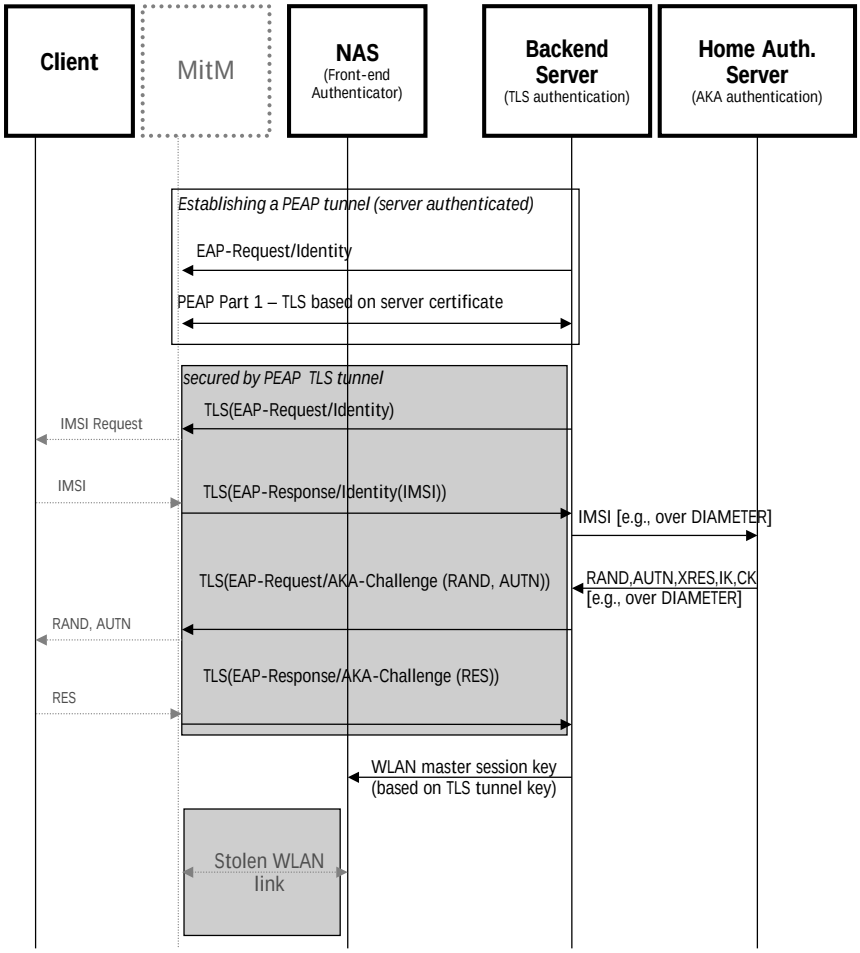

Fig. 4. Man-in-the-Middle in PEAP, e.g., with EAP AKA

2. S is the client's long-term authentication key and is accessible for derivation of session keys directly, outside the authentication protocol.

A typical example of a protocol of the first type is EAP AKA. Figure 3 illustrates how EAP AKA can be used within the PEAP protocol. The AKA quintuplets contain 256 bits of secret session keys which would be readily available for session key derivation by the backend server. These AKA session keys are also available in the client's device. There are several examples of existing client authentication protocols that support session key derivation. For such protocols, no changes are required to provide the necessary cryptographic binding.

In the second case, it is necessary to specify an additional key derivation application that has direct access to client's authentication key. This would be in addition to the existing authentication protocol and is to be judged separately for each case.

Let K be the ultimate session key. In PEAP and EAP-TTLS the session key K is the master key that finally becomes available to the local backend server (access server), which uses K to derive further session keys. Let T denote the master key that is used to derive the secret keys for the protection tunnel. For example, the TLS master key derived in the TLS handshake of PEAP is a typical example of T.

We achieve the binding between the inner and outer protocols by binding S and K. On the network side some entity, let us call it "binding agent", is responsible for collecting the secret key information S and T and creating the binding value. Typically the binding agent is either the local backend server or the home authentication server. If S is a long term authentication key of the client, then the binding agent is preferably co-located with client's home authentication server, to avoid transfer of S across the network.

As mentioned, there are two ways of using S to achieve the necessary binding of S to K. In implicit binding, the binding is established directly by taking S in addition to T as input to the session key computation. The binding agent and the client each compute its copy of the session key K from S and T using a pseudo random function suitable for key derivation. The binding agent distributes the session key to the network entities that need to use it for further communication with the client. This provides implicit authentication of the client.

In explicit binding we make use of a cryptographic check value to verify that the client who is in possession of T is also in possession of S. The binding agent and the client each compute its copy of a verification value V from S and T using a cryptographic hash function or a message authentication function. They then transfer their verification values to some network entity responsible for comparing the two verification values. If they are equal, the client is granted access to the network service. The comparing entity can be the backend server or the home authentication server. This provides explicit authentication of the client. If explicit binding is used then the session key K can be based on S or T or both.

For both the implicit and explicit binding mechanisms we assume that the process to construct the tunnel secret T is contributory. However, it is not necessary that the server is explicitly authenticated by the client in the tunnel establishment. It is also possible for the client to rely on the home server for the server authentication. Then the binding of S to K provides server authentication also in the cases where the client neglects the result of the server authentication during tunnel setup, or an anonymous tunnel is used.

Explicit binding is always necessary in case the session key K is based solely on the tunnel secret T. A variation of explicit binding works as follows. In the computation of the verification value V, some tunnel-indication data D is used instead of T. For example, D can be an entire protocol message sent by the client, or the addresses of the parties or just a text string unique to the application. This variation is useful in cases where access to T is not available because of implementation constraints (e.g., a black-box implementation of TLS will not allow the TLS master key to be accessible outside TLS usage). In typical scenarios, D does not specify T in a unique manner. Then this approach is weaker than the normal explicit binding, described earlier, because it relies on the client carrying out server authentication properly during tunnel setup.

It is also possible to implement both implicit and explicit binding. In such a case the explicit binding verification acts as a key confirmation for the agreed session key.

5 Weak authentication Protocols

Consider a client authentication protocol based on weak secrets like passwords. Suppose further that the protocol does not use any modern techniques to protect the weak secret against guessing attacks. Many of the tunnelling protocols described earlier, e.g., the PIC protocol, are particularly intended to address such weak authentication protocols, which are insecure without added protective tunnelling. One may ask what is the impact of the cryptographic binding described in section 4 on such a case. Does the binding possibly improve the overall security? It is clear that if the client is unable to perform server authentication for the outer tunnel protocol, then the binding does not improve the security: an attacker could masquerade as the server in protocol run, and use the information gained in a dictionary attack on the password.

However, all the examples we considered in section 2 do assume that the client can perform server authentication for the tunnel protocol. In this case, the attacker cannot pretend to be the server end of the tunnel towards the client. If the attacker is a passive eavesdropper, he cannot perform the dictionary attack on the binding because the binding includes the outer tunnel key which is assumed to have sufficient entropy. If the attacker pretends to be the client towards the server and if the server does the binding first (e.g., if the server sends the binding verification value to the client, or if it starts using the key K or S to encrypt or authenticate messages), then he can perform a dictionary attack. But this is easily prevented by requiring that the client does the binding first and demonstrate knowledge of S and T to the server. If this step fails, the server must terminate the session.

Thus, the cryptographic binding does not *reduce* the security of a weak authentication protocol. However, to be fully secure, weak authentication protocols used with server authenticated tunnels must satisfy two conditions:

- A1 *Correct server authentication*: the client **must** perform server authentication correctly, and
- A2 *No mixing of authentication modes*: if a client uses tunnelled authentication, it **must not** use the same authentication protocol outside secure tunnels.

Assumption A2 is unavoidable for weak authentication protocols. But it clearly diminishes the capability of the tunnelling protocol to leverage the advantages of

already deployed legacy authentication protocols. Well-designed authentication protocols do not need assumption A2.

If assumption A2 can be made in general, cryptographic binding is not necessary, but the legacy advantages are lost. On the other hand, cryptographic binding is not applicable to the legacy authentication protocols that do not yield a key suitable for cryptographic binding.

Generic tunnelling protocols, by definition, should be able to work with all types of authentication protocols while making as few assumptions about them as possible. Therefore a generic tunnelling protocol must not take A2 as a blanket assumption to be imposed on all authentication protocols. Instead it should allow the possibility of using the type of cryptographic binding described in section 4 where appropriate. This way, the tunnelling protocol can be secure, generic (supporting both weak and strong authentication protocols), and non-invasive (avoiding unnecessary restrictions on strong authentication protocols).

6 Impact and outlook

The attack is applicable to a number of proposed protocols, including PIC, IKEv2, XAUTH, PEAP, EAP TTLS and POTLS [1]. These are not academic toy protocols, but protocols with a good prospect of being deployed and used widely. For IKEv2 it concerns, in particular, the Secure Legacy Authentication (SLA) protocol, which is intended to be included in the main body of IKEv2. The conclusion from the discussion at the IPSEC list [24] was that an explicit cryptographic binding for SLA was to be specified. The binding is mandated in case the legacy authentication method provides a shared secret. This change has already been included in a more recent version of the IKEv2 draft [10].

The sheer number of different protocol combinations where the same MitM problem has manifested is surprising. One reason is the lack of easily accessible tools for specifying and verifying security properties of protocols. The security of protocol composability is still an emerging area of research. In [18], Meadows points out the current status of research as well as other practical examples of security failures arising from protocol composition. Also, even though the security protocols research community has been working on protocol verification for decades, the current tools and methodologies require significant expertise, and are not yet ready for use by people who are not actively doing protocol verification research. It is only recently that experts from the security protocols community have started analyzing draft specifications from open standardization bodies [18]. As Meadows points out, the possibility to analyze draft protocols before they become standards is an important opportunity for researchers in this area.

A second possible reason is that, in all of the cases examined, the legacy authentication protocol was a framework, rather than a concrete protocol. This means that the properties of individual authentication protocols are not readily noticeable to the designers of the tunnelling approach. A particular problem is created by the large variety of legacy protocols which differ in the level of security provided: e.g., some of them provide strong mutual authentication, and others are based on a user selected password. It is not clear if it is useful or even possible to accommodate all different protocols within the same tunnelling framework.

7 Conclusion

In this paper we have shown that when a client authentication protocol is tunnelled within another protocol, it is necessary for each endpoint to demonstrate that it has participated in both protocols within the authentication exchange. If this is not demonstrated then the tunnelled authentication protocol is vulnerable to a Man-in-the-Middle attack.
We have also shown that the required demonstration can be provided in an implicit or explicit way in a form of a cryptographic binding between the tunnel protocol and the authentication protocol. In our proposals the binding facility is implemented in the outer tunnel protocol. It requires the authentication protocol to provide some secret key values for the use of the binding. This approach is preferred since it requires minimal or no changes to the authentication protocols. It allows for flexible and secure usage of an authentication protocol in multiple authentication environments without the authentication protocol being aware of the specific environment.
The cryptographic binding proposed in this paper does not diminish the security of the tunnelled protocols in any case. If the inner authentication protocol is a weak authentication protocol based on a weak client secret, the tunnel must be constructed based on server authentication, and the client should not use the same secret in different environments. Otherwise, the protocol is vulnerable to dictionary attacks, with or without cryptographic binding. Strong authentication methods are not vulnerable to dictionary attacks, and hence should not be restricted to tunnelled environments only. Hence the need for the type of binding described in this paper.

8 Current developments

We pointed out the MitM problem to the authors of some of the tunnelling proposals in early October, 2002. Since then a number of positive developments have taken place. The IETF EAP working group has recognized the problem of securely binding a sequence of authentication methods together. It is now marked as an open issue to be solved [6]. Another draft [21] describes the type of solutions prescribed in this paper. Designers of some of the affected protocols have published new versions of their specifications: for example, the new version of the PEAP draft specification [2] adopts the techniques specified in [21] to protect against MitM attacks. Other affected specifications, such as EAP SIM GMM authentication [7] and PIC [23] have been withdrawn altogether.
We therefore expect that in due course of time the problem will be solved in all the identified specifications. However, it requires significant analysis and development for each protocol. In each case one must decide at which protocol message the binding is implemented, which parties shall create the binding, and how to derive the shared secrets used for the binding. Different decisions result in different protocol properties.
Client authentication protocols may take additional safeguards to protect against MitM. For example, in a newer version of the EAP AKA specification [4], the authentication response must include RES as well as a message authentication code on RES. Thus the attack described in Figure 4 is no longer possible. However, the MitM can still pretend to be a backend server towards the victim. Therefore, if

mixed mode usage is unavoidable, then ultimately the tunnelling protocols have to be fixed.
As we mentioned already, the problem is not limited to EAP only. For example, HTTP Digest AKA specification [19] effectively uses the HTTP authentication protocol [11] as a framework. The standard combination of HTTP authentication with TLS is thus rendered insecure when the underlying HTTP authentication is HTTP Digest AKA, despite the fact that HTTP Digest AKA is a secure protocol providing mutual authentication and strong session key agreement.

Acknowledgments:
We thank Henry Haverinen for many interesting and useful discussions on this and other related topics. We also thank Jan-Erik Ekberg, Dan Forsberg, Pasi Eronen, Philip Ginzboorg, Tao Haukka, David Jablon, Hugo Krawczyk, Jose Puthenkulam, Jarno Rajahalme, Heikki Riittinen and the participants of the IST-SHAMAN project for discussions and valuable feedback on our ideas.

References

1. Bernard Aboba. Review of man-in-the-middle problem statement draft. Message to IETF saag mailing list, January 2003. `http://jis.mit.edu/pipermail/saag/2003q1/000684.html`.
2. H. Andersson, S. Josefsson, Glen Zorn, Dan Simon, and Ashwin Palekar. Protected EAP Protocol (PEAP), September 2002. IETF personal draft `draft-josefsson-pppext-eap-tls-eap-05.txt`.
3. J. Arkko and H. Haverinen. EAP AKA Authentication, June 2002. IETF personal draft `draft-arkko-pppext-eap-aka-04.txt`.
4. J. Arkko and H. Haverinen. EAP AKA Authentication, January 2003. IETF personal draft `draft-arkko-pppext-eap-aka-08.txt`.
5. N. Asokan, Valtteri Niemi, and Kaisa Nyberg. Man-in-the-middle in tunneled authentication protocols. Technical Report 2002/163, IACR ePrint archive, October 2002. `http://eprint.iacr.org/2002/163/`.
6. L. Blunk, J. Vollbrecht, and Bernard Aboba. Extensible Authentication Protocol (EAP), October 2002. IETF *pppext* working group draft `draft-ietf-pppext-rfc2284bis-07.txt`.
7. Adrian Buckley, Prasanna Satarasinghe, Vladmir Alperovich, Jose Puthenkulam, Jesse Walker, and Victor Lortz. EAP SIM GMM Authentication, August 2002. IETF personal draft `draft-buckley-pppext-eap-sim-gmm-00.txt`.
8. Pat Calhoun et al. Diameter Base Protocol, December 2002. IETF *aaa* working group draft `draft-ietf-aaa-diameter-17.txt`.
9. T. Dierks and C. Allen. The TLS Protocol Version 1.0, January 1999. IETF RFC 2246.
10. Charlie Kaufman (Editor). Internet Key Exchange (IKEv2) Protocol, February 2003. IETF *ipsec* working group draft `draft-ietf-ipsec-ikev2-05.txt`.
11. J. Franks et al. HTTP Authentication: Basic and Digest Access Authentication, June 1999. IETF RFC 2617.
12. Paul Funk and Simon Blake-Wilson. EAP Tunneled TLS Authentication Protocol (EAP-TTLS), February 2002. IETF *pppext* working group draft `draft-ietf-pppext-eap-ttls-01.txt` (expired).

13. Dan Harkins, Derrel Piper, and Paul Hoffman. Secure Legacy Authentication (SLA) for IKEv2, December 2002. IETF personal draft `draft-hoffman-sla-00.txt`.
14. H. Haverinen and J. Salowey. EAP SIM Authentication, October 2002. IETF personal draft `draft-haverinen-pppext-eap-sim-06.txt`.
15. S. Josefsson, A. Palekar, D. Simon, and G. Zorn. Protected EAP Protocol (PEAP), March 2003. IETF personal draft `draft-josefsson-pppext-eap-tls-eap-06.txt`.
16. Heikki Kaaranen, Siamäk Naghian, Lauri Laitinen, Ari Ahtiainen, and Valtteri Niemi. *UMTS Networks: Architecture, Mobility and Services.* John Wiley & Sons, 2001.
17. J. Vollbrecht L. Blunk. PPP Extensible Authentication Protocol (EAP), March 1998. IETF RFC 2284.
18. Catherine Meadows. Formal methods for cryptographic protocol analysis: emerging issues and trends. *IEEE Journal on Selected Areas in Communications*, 21(1):44–54, January 2003.
19. Aki Niemi, Jari Arkko, and Vesa Torvinen. Hypertext transfer protocol (http) digest authentication using authentication and key agreement (aka). IETF RFC 3310, September 2002.
20. Yoshihiro Ohba, Shinichi Baba, and Subir Das. PANA over TLS (POTLS), September 2002. IETF personal draft `draft-ohba-pana-potls-00.txt`.
21. Jose Puthenkulam, Victor Lortz, Ashwin Palekar, Dan Simon, and Bernard Aboba. The compound authentication binding problem, March 2003. IETF personal draft `draft-puthenkulam-eap-binding-02.txt`.
22. C. Righney et al. Remote Authentication Dial In User Service (RADIUS), June 2000. IETF RFC 2865.
23. Y. Sheffer, H. Krawczyk, and Bernard Aboba. PIC, A Pre-IKE Credential Provisioning Protocol, October 2002. IETF *ipsra* working group draft `draft-ietf-ipsra-pic-06.txt`.
24. IETF IPsec working group. Secure legacy authentication for IKEv2. Discussion thread on the IPSec mailing list. `http://www.vpnc.org/ietf-ipsec/mail-archive/threads.html#02763`.