Identity-Based Protocol Design Patterns for Machine-to-Machine Secure Channels

Francisco Corella
Pomcor
Email: fcorella@pomcor.com

Karen P. Lewison
Pomcor
Email: kplewison@pomcor.com

***Abstract*—In the classical design pattern for secure channel protocols, there is a first phase where the endpoints establish a shared secret and one or both endpoints authenticate by presenting a certificate and demonstrating knowledge of the associated private key; and a second phase where application data traffic between the endpoints is protected using symmetric keys for encryption and authentication derived from the shared secret. This classical pattern emerged in the nineties, when most secure channels were intended for human-to-machine (H2M) communications and the latency caused by roundtrips and transmission of bulky certificate chains resulted in no more than inconvenience to the human. In machine-to-machine (M2M) communication, on the other hand, excessive latency may be unacceptable for safety-critical and other applications. We describe a range of alternative protocol design patterns that use identity-based encryption to eliminate roundtrips and certificate transmission. These patterns can be used in the design of new secure channel protocols or retrofitted into existing protocols.**

I. Introduction

A. The Classical Pattern for Secure Channel Protocol Design

A secure channel transmits data securely between two endpoints over a potentially compromised network, providing security services that typically include authentication of one or both endpoints, data confidentiality, and data authentication.

The most widely used secure channel protocols make use of the same protocol design pattern in at least some of their variants. We shall refer to this pattern as *the classical pattern*. In the classical pattern, a protocol includes two phases. In the first phase, the endpoints establish a shared secret and one or both endpoints authenticate by presenting a certificate issued by a certificate authority (CA) and demonstrating knowledge of the associated private key. In the second phase, application data traffic between the endpoints is protected using symmetric keys for data encryption and authentication derived from the shared secret. This design pattern is notably used by TLS [1] and IPsec [2]. It is also an option in SSH [3].

B. Drawbacks of the Classical Pattern

The classical pattern has two usability drawbacks.

First, the first phase requires one or more roundtrips between the endpoints before protected traffic starts flowing, and those roundtrips add latency to the process of establishing a secure channel. In TLS, for example, where the first phase is known as the handshake, a full handshake requires two roundtrips, while an abbreviated handshake for resuming a previously established "session" requires one roundtrip. IPsec is a suite of protocols. The first phase is specified by the Internet Key Exchange (IKE) protocol [4] and compromises two or more roundtrips. An extension [5] provides a session resumption mechanism that reduces the number of roundtrips, similar to the abbreviated handshake of TLS.

Second, the first phase also requires transmission of certificates between the endpoints. Certificates are bulky: for example, as of this writing Bank of America uses a 2244B TLS certificate, backed by an 2136B intermediate CA certificate, for a total certificate chain size of 4380 bytes. If both endpoints authenticate with certificate chains the total bandwidth taken up by certificate transmission could exceed 8KB.

This second drawback compounds the first, because certificate transmission on a bandwidth-constrained network can result in congestion, which in turn results in increased latency.

The classical design pattern emerged during the nineties, at a time when most secure channels supported human-to-machine (H2M) interactions. In such channels, the above drawbacks result in an inconvenience to the human. In some cases that inconvenience may be severe, e.g. when a user tries to access the web over a geostationary satellite, in which case the speed of light imposes a roundtrip delay of more than half a second, and bandwidth is scarce; or when bandwidth-constrained radio links are used. (Presumably, military applications that rely heavily on radio and satellite links make little use of TLS or IPsec.) But in most cases, the inconvenience to the human is negligible or tolerable.

The impact of the drawbacks, however, is very different in the case of machine-to-machine (M2M) communications. First, some M2M networks are highly bandwidth-constrained. For example, ZigBee transmission rates vary from 20 to 250 kilobits/second [6]. Second, in M2M communications latency is not a matter of inconvenience. Some M2M applications, including safety-critical ones, are latency-intolerant. For example, at a speed of 70mph, a car travels 10 feet in 100ms; a 100ms increase in reaction time to the latency of establishing a secure channel may make it impossible to avoid a collision that could otherwise have been avoided.

To appear in Proceedings of M2Msec'14 ©2014 IEEE

C. Mitigations of the Drawbacks

The drawbacks of the classical pattern are well known and many techniques have been used to mitigate them within the framework of different protocols. Some protocols provide the option of using preshared keys to avoid the transmission of certificates, at the cost of introducing a shared key distribution problem. Some protocols allow the use of an uncertified key pair instead of a certificate and its associated private key for authentication of one or both endpoints. Uncertified key pairs are often used, for example, in SSH [3]. A bare public key is then transmitted instead of a certificate chain, saving much bandwidth. Uncertified key pairs are practical and may be preferable to certificates when the two end-points have a longstanding relationship, as is the case for SSH; but they are impractical otherwise.

Several techniques have been implemented or proposed specifically to reduce the number of roundtrips and the transmission of certificates during the TLS handshake.

The above-mentioned abbreviated handshake, which is part of the core TLS protocol [1, §7.3] eliminates the transmission of certificates and reduces the number of roundtrips from two to one. It can be used when the TLS client and server have cached parameters of a previous connection. A protocol extension [7] allows the server to save the parameters in the client (encrypted) to save space in the server cache and thus increase the chances that a session resumption request by the TLS client can be satisfied. It has been reported that "the observed attempted resume rate at Google is only about 50%" [8].

The Fast-track mechanism [9] relies on the client caching server parameters of an initial connection without requiring the server to remember client parameters. It eliminates the transmission of the server certificate in subsequent connections, and saves one roundtrip by letting the client send application data with the third handshake flow. Fast-track was implemented but not deployed.

The False Start mechanism [10] saves one roundtrip by letting the client send protected application data before receiving the *Finished* message from the server that concludes the handshake and authenticates the server to the client. This is secure because the data can only be decrypted by the authentic server. False Start was implemented by Google on the Chrome browser without specifying a TLS extension, taking advantage of the fact that most server implementations accept early data; but it had be discontinued as no reliable way could be found of detecting servers that did not accept early data in order to fall back on standard protocol behavior for those servers [11].

The Snap Start mechanism [12] saves both roundtrips and transmission of the server certificate when the client is able to predict the server's messages based on previous interactions. It has not been deployed.

D. Alternative Patterns

The above mitigations alleviate the usability drawbacks of the classical pattern, but at the cost of increased complexity and without fully eliminating them. It is obviously desirable to find alternative patterns that do not suffer from the same drawbacks and do not require mitigations.

The previously mentioned idea of using uncertified key pairs as credentials instead of certificates and their associated private keys, besides being used as an option in some protocols, has been used as a stand-alone pattern in the context of the Simple Public Key Infrastructure (SPKI) [13], the Host Identity Protocol (HIP) [14], and the Privilege Management Protocol [15]. But this pattern still requires one or more roundtrips, and transmission of a public key. A public key is less bulky than a certificate chain but it may still contribute to congestion in bandwidth-constrained networks.

On the other hand, a powerful but little used cryptographic technique, identity-based (ID-based) encryption, can fully eliminate the usability drawbacks. A step in that direction was recently made by Mulkey and Kar [16], who proposed a design of a wireless security protocol using ID-based encryption for authentication and key exchange in order to eliminate the transmission of certificates. However their design requires three roundtrips for key exchange and unilateral authentication. In this paper we describe a range of ID-based protocol design patterns that are generally applicable to a broad range of secure channel protocols and eliminate roundtrips as well as certificate transmission. In the rest of the paper we briefly explain the concept of ID-based encryption, describe the ID-based design patterns, and discuss techniques that enable the large scale deployment of ID-based secure channels.

II. ID-Based Encryption

ID-based cryptography was pioneered in 1984 by Adi Shamir, who described an ID-based cryptosystem for digital signature [17]. Many other ID-based cryptosystems have been proposed for digital signature, encryption, and key agreement; a survey of a large number of ID-based cryptosystems for encryption can be found in [18].

In an ID-based cryptosystem the public key of an entity is computed from (or, equivalently, consists of) an identifier of the entity and the public key of a trusted party, while the private key is computed from the same identifier and the private key of the trusted party. The trusted party computes the private key and conveys it to the entity through a secure out-of-band channel; for that reason the trusted party is called the *private key generator* (PKG). The PKG plays a role equivalent to that of a CA in a public key infrastructure (PKI).

Generally speaking, having the private key generated by the PKG and sent to the entity is less secure than having the key pair generated by the entity as is the case in a traditional PKI, because of the risk of misuse of the private key by the PKG, or by an adversary who captures it by breaching the security of the out-of-band channel. However, when the private key is used for authentication rather than non-repudiation, the risk to be considered is impersonation rather than repudiation, and in a traditional PKI there are equivalent risks of impersonation of an entity by a CA (which may issue itself a certificate) or by an adversary who modifies a certificate signing request to substitute the adversary's own public key for the entity's.

An ID-based key pair can be made to expire by augmenting the identity from which the public key is computed with an expiration time. This requires a means of securely delivering a new private key to the entity before the old one expires. We further propose a means of revoking an ID-based key pair, by augmenting the identity with a revocation count and making the augmented identity retrievable from a secure repository or service (not necessarily the PKG) such as DNSSEC. Expiration and revocation can be combined.

III. ID-based Protocol Design Patterns for Secure Channels

We now describe four ID-based protocol design patterns for secure channel protocols between an initiator and a responder: a basic pattern where only the responder authenticates; a mutual authentication pattern where both the initiator and the responder authenticate; a pattern that provides forward secrecy with mutual authentication; and a pattern that provides forward secrecy with responder-only authentication. We leave to the reader the details of patterns where only the initiator is authenticated. For simplicity, in all of the following patterns it is assumed that protocol parameters are fixed and need not be negotiated between the initiator and the responder.

A. Secure Channel with Responder Authentication

The first pattern, shown in Figure 1, comprises the following steps.

Step 1: The initiator computes the responder's public key (resp-pubkey) from the responder's identity and the public key of the PKG, generates a a random high-entropy nonce (init-nonce), and derives traffic protection keys (keys1-i2r) from the nonce using a key derivation function such as HKDF [19]. In general, a complete set of traffic protection keys comprises a subset of keys for protecting application data sent from the initiator to the responder and a subset for protecting data traveling from the responder to the initiator; each subset comprises an encryption key and an authentication key, or a single key for authenticated encryption. However in the case of the first set of traffic protection keys (keys1), only the initiator-to-responder subset (keys1-i2r) is needed.

Step 2: The initiator sends its nonce to the responder, encrypted under the responder's public key. It also starts sending application data in the same flow, protected by, or, synonymously, *wrapped under* keys1-i2r. Since the responder does not contribute randomness to the generation of those protection keys, an adversary can replay this flow. However the initiator should be allowed to repeat the first flow if a response is not received promptly, hence the responder should tolerate replay of the first flow.

Step 3: The responder decrypts the initiator's nonce, derives keys1-i2r from the nonce, and decrypts the application data sent by the initiator. Then it generates its own nonce (resp-nonce) and derives a second set of traffic protection keys (keys2) using the same key derivation function, with the two nonces and the identity of the responder as input keying material. The purpose of including the identity of the responder is to avoid an *unknown key share* (UKS) flaw like the one used in the recently discovered Triple Handshake Attack against TLS [20].

Step 4: The responder sends its nonce to the initiator in the clear. After that, all messages from the responder to the initiator are protected by the responder-to-initiator subset of the second set of traffic protection keys (keys2-r2i). In particular, in the same flow, the responder returns the initiator's nonce and sends application data (data2), both wrapped under keys-r2i.

Step 5: The initiator derives the second set of traffic protection keys from the two nonces and the responder's identity. Then it decrypts and checks its nonce, returned by the responder; a successful check authenticates the responder and verifies agreement on keys2-r2i. Then it decrypts the application data sent by the responder.

Step 6: From now on all messages from the initiator to the responder are protected by the second set of traffic protection keys. In step 6, the initiator returns the responder's nonce and sends application data (data3), both wrapped under keys2-i2r.

Step 7: The responder decrypts and checks its nonce, returned by the initiator; a successful check verifies agreement on keys2-i2r. Then it decrypts the application data sent by the initiator.

After step 7, the initiator and the responder can continue to exchange application data protected by the second set of traffic protection keys.

B. Secure Channel with Mutual Authentication

The second pattern, which adds initiator authentication to security services provided by the secure channel protocol, is shown in Figure 2. It differs from the basic pattern as follows. At step 3 the responder further computes the initiator's public key (init-pubkey) from the initiator's identity and the public key of the PKG. At step 4 it uses it to encrypt its nonce, which the initiator decrypts at step 5. At step 7 a successful check of the responder's nonce returned by the initiator authenticates the initiator, in addition to verifying agreement on keys2-i2r.

C. Mutual Authentication plus Forward Secrecy

Forward secrecy cannot be provided for application data sent in the first flow, but it can be provided for subsequent flows.[1] To that purpose we use ephemeral Diffie-Hellman (EDH) key agreement, either traditional or based on an elliptic curve [22]. The EDH domain parameters [22, §5.5] may be generated by a trusted party and may be common to all entities or to a subset of the entities, or they may be generated by the client. When generated by the initiator we consider them to

[1] The idea of upgrading a connection that does not provide forward secrecy for the initial flow so that it does provide it for subsequent flows has been independently suggested in the design document of QUIC [21], a Google protocol that multiplexes data streams over UDP.

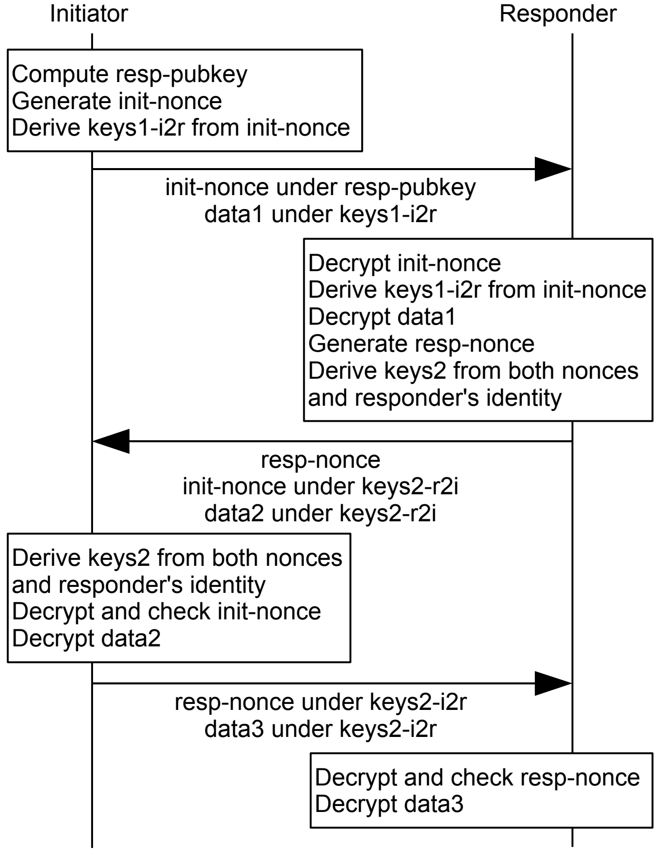
Initiator
Responder
Compute resp-pubkey
Generate init-nonce
Derive keys1-i2r from init-nonce
init-nonce under resp-pubkey
data1 under keys1-i2r
Decrypt init-nonce
Derive keys1-i2r from init-nonce
Decrypt data1
Generate resp-nonce
Derive keys2 from both nonces
and responder's identity
resp-nonce
init-nonce under keys2-r2i
data2 under keys2-r2i
Derive keys2 from both nonces
and responder's identity
Decrypt and check init-nonce
Decrypt data2
resp-nonce under keys2-i2r
data3 under keys2-i2r
Decrypt and check resp-nonce
Decrypt data3

Fig. 1. Secure channel with responder authentication

be part of the EDH public key. Domain parameters generated by the initiator must be validated by the responder.

The third pattern, which provides forward secrecy from the second flow in addition to mutual authentication, is shown in Figure 3. It differs from the second pattern as follows. At step 1, the initiator generates its EDH key pair (init-edh-keypair). At step 2, the initiator sends its EDH public key (init-edh-pubkey) in the clear. (The EDH public key init-edh-pubkey should not be confused with the initiator's ID-based public key, which is called init-pubkey.) At step 3, the responder generates its EDH key pair (resp-edh-keypair), computes the EDH shared secret (edh-secret), and derives the second set of traffic protection keys (keys2) from the the EDH shared secret instead of deriving it from the nonces and the responder's identity. (It is not necessary to include the responders's identity in the derivation of keys2 provided that valid EDH domain parameters are used.) At step 5, the initiator derives the second set of traffic protection keys from the the EDH shared secret instead of deriving it from the nonces and the responder's identity.

D. Forward Secrecy with Responder-only Authentication

The fourth pattern, shown in Figure 4, is a simplification of the third one. At step 4 the responder sends its nonce in the clear instead of sending it encrypted under the initiator's ID-based public key, and at step 5 the initiator does not need to decrypt the responder's nonce.

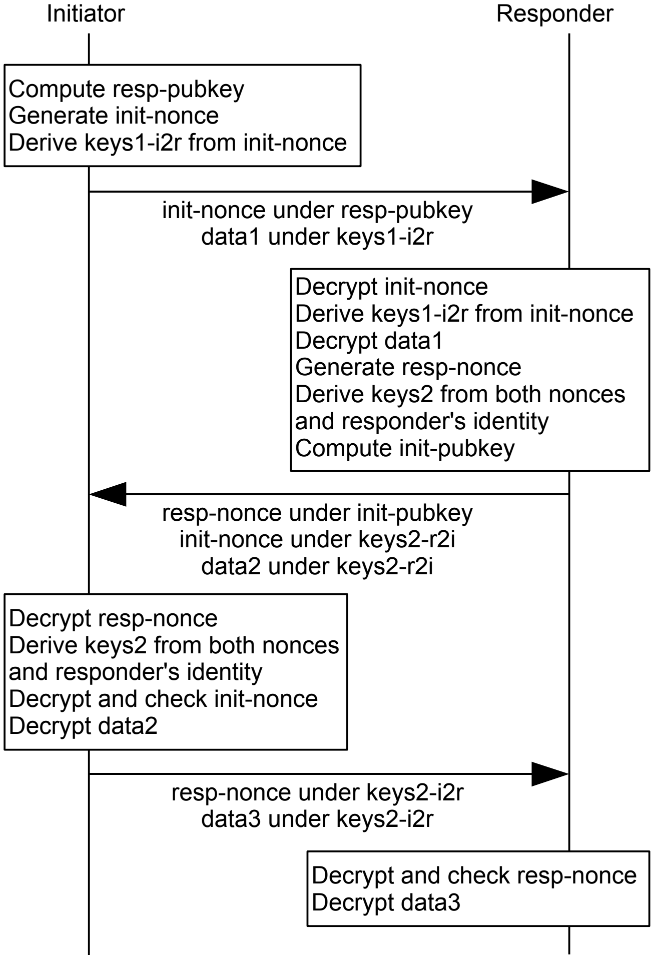
Initiator
Responder
Compute resp-pubkey
Generate init-nonce
Derive keys1-i2r from init-nonce
init-nonce under resp-pubkey
data1 under keys1-i2r
Decrypt init-nonce
Derive keys1-i2r from init-nonce
Decrypt data1
Generate resp-nonce
Derive keys2 from both nonces
and responder's identity
Compute init-pubkey
resp-nonce under init-pubkey
init-nonce under keys2-r2i
data2 under keys2-r2i
Decrypt resp-nonce
Derive keys2 from both nonces
and responder's identity
Decrypt and check init-nonce
Decrypt data2
resp-nonce under keys2-i2r
data3 under keys2-i2r
Decrypt and check resp-nonce
Decrypt data3

Fig. 2. Secure channel with mutual authentication

IV. Large Scale Deployments with Multiple PKGs

Since each communicating entity in a deployment of an ID-based secure channel protocol receives its private key from a PKG, multiple PKGs are needed for a large deployment; this is especially necessary if entity identities include short term expiration times, so that private keys have to be renewed frequently. But then each entity must know which PKGs have issued the private keys of the entities it communicates with, and the public keys of those PKGs.

A *hierarchical ID-based encryption* (HIBE) cryptosystem such as the one described in [23] (see [18] for references to others) can be used to address this problem. In a HIBE cryptosystem there is a tree of PKGs. Each node in the tree issues private keys to its children, and leaf nodes (as well as, possibly, other nodes) issue private keys to communicating entities. Each PKG other than the root has an identity. Each communicating entity has an extended identity comprising: a base identity, which uniquely identifies the entity within

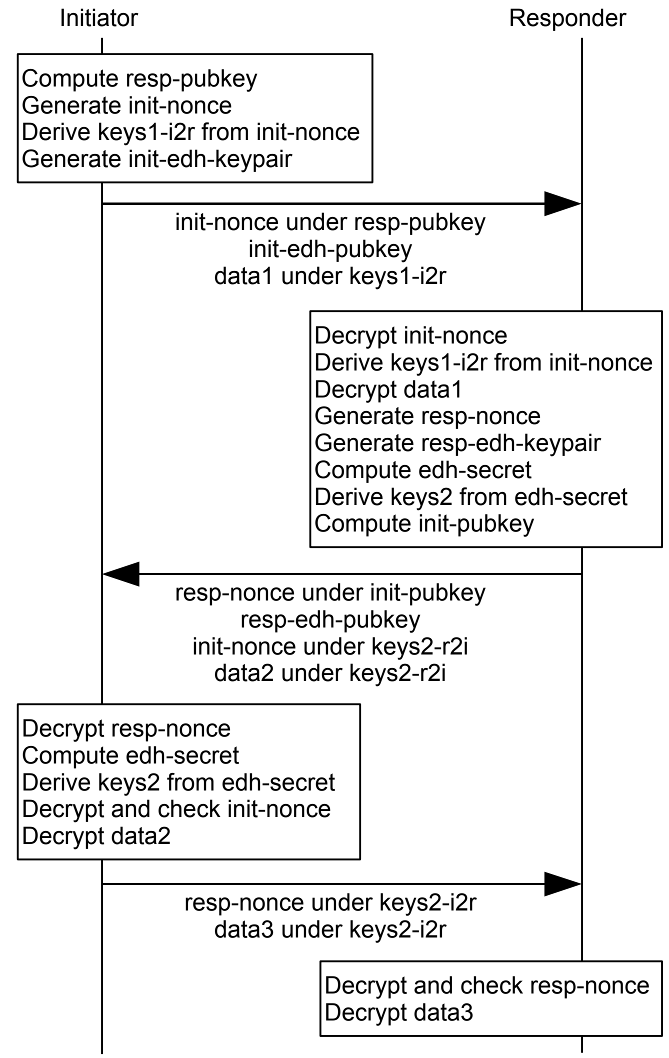
Initiator
Responder
Compute resp-pubkey
Generate init-nonce
Derive keys1-i2r from init-nonce
Generate init-edh-keypair
init-nonce under resp-pubkey
init-edh-pubkey
data1 under keys1-i2r
Decrypt init-nonce
Derive keys1-i2r from init-nonce
Decrypt data1
Generate resp-nonce
Generate resp-edh-keypair
Compute edh-secret
Derive keys2 from edh-secret
Compute init-pubkey
resp-nonce under init-pubkey
resp-edh-pubkey
init-nonce under keys2-r2i
data2 under keys2-r2i
Decrypt resp-nonce
Compute edh-secret
Derive keys2 from edh-secret
Decrypt and check init-nonce
Decrypt data2
resp-nonce under keys2-i2r
data3 under keys2-i2r
Decrypt and check resp-nonce
Decrypt data3

Fig. 3. Secure channel with mutual authentication and forward secrecy

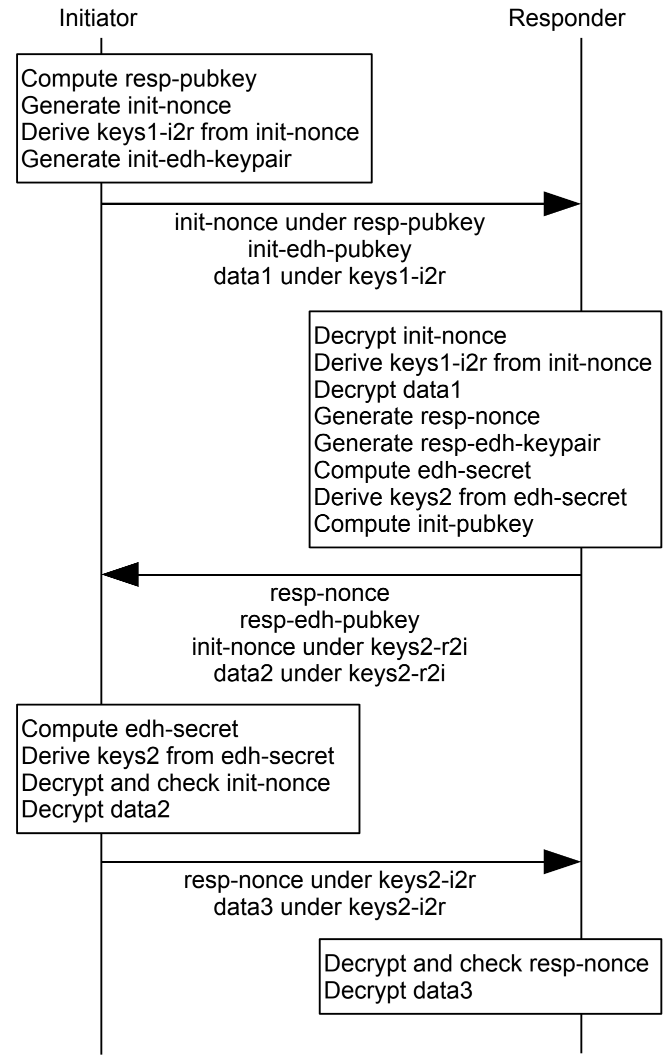
Initiator
Responder
Compute resp-pubkey
Generate init-nonce
Derive keys1-i2r from init-nonce
Generate init-edh-keypair
init-nonce under resp-pubkey
init-edh-pubkey
data1 under keys1-i2r
Decrypt init-nonce
Derive keys1-i2r from init-nonce
Decrypt data1
Generate resp-nonce
Generate resp-edh-keypair
Compute edh-secret
Derive keys2 from edh-secret
Compute init-pubkey
resp-nonce
resp-edh-pubkey
init-nonce under keys2-r2i
data2 under keys2-r2i
Compute edh-secret
Derive keys2 from edh-secret
Decrypt and check init-nonce
Decrypt data2
resp-nonce under keys2-i2r
data3 under keys2-i2r
Decrypt and check resp-nonce
Decrypt data3

Fig. 4. Responder-only authentication with forward secrecy

the cryptosystem; and a PKG identity chain that lists the identities of the PKGs along the path within the tree from the PKG that issues the private key of the entity up to, but not including, the root of the PKG tree. The public key of an entity is computed from (or, equivalently, consists of) the public key of the root of the tree and the extended identity of the entity. Thus each entity only needs to know the public key of the root PKG and the extended identities of the entities with which it communicates.

While a HIBE cryptosystem may solve the problem for a large deployment, more may be needed for a *global deployment*. Just as multiple root CAs are needed for the global PKI that provides certificates to web servers, multiple root PKGs may be needed for a global deployment of an ID-based secure channel protocol. To enable such global deployments we introduce the concept of a Multi-Root HIBE (MR-HIBE) cryptosystem. A MR-HIBE cryptosystem is an extension of an ordinary HIBE cryptosystem to accommodate a forest of PKGs. Each root of the forest has its own public key. Every PKG in the forest, including the roots, has an identity. Each entity has an extended identity that comprises a base identity and a PKG identity chain. The PKG identity chain lists the identities of the PKGs along the path within the forest from the PKG that issues its private key to the entity up to, *and including*, a root of the PKG forest. The public key of an entity is computed from (or, equivalently, consists of) the extended identity of the entity and the public key of the root PKG whose identity appears at the end of the PKG identity chain. Each entity stores the public keys of the root PKGs that it trusts, just as a web browser stores the certificates of the root CAs that it trusts.

A difficulty remains. An entity that needs to establish a secure channel to communicate with a target entity may know the base identity of the target entity but not its extended identity. It must therefore retrieve the PKG identity chain of the target entity. Retrieving a PKG identity chain is akin to retrieving a certificate chain in a PKI, but each certificate in a certificate chain is hundreds or thousands of bytes long, whereas each identity in a PKG identity chain is only a few bytes long. This makes a difference: in an Internet deployment, the initiator of a secure channel could obtain the PKG identity chain of the responder from the Domain Name System (DNS) with the same query that it uses to look up the responder's IP address, avoiding a roundtrip. By contrast, a web browser could not reliably retrieve the certificate chain of a web server from the DNS because DNS responses are traditionally carried by 512 byte UDP datagrams, and buggy implementations of DNS resolvers may fail on large responses.

Obtaining the PKG identity chain of a target entity from a third party service such as the DNS rather than from the target party itself mitigates the risk of impersonation of the target by an attacker who uses a compromised PKG different than the PKG used by the target to obtain a private key for the target's identity. (The corresponding risk in a PKI setting is well known; it was the topic of a recent NIST workshop [24].) If the third party service is trusted and secure, e.g. if DNSSEC is available, the risk is eliminated. (In the PKI setting, the DANE protocol [25] eliminates the risk by obtaining a hash of a server certificate from DNSSEC, if DNSSEC is available.)

In a HIBE or MR-HIBE cryptosystem, the base identity of an entity may not be atomic. It may be an augmented identity derived from an atomic identity by appending an expiration time and/or a revocation count, as described above at the end of Section II. The expiration time may be known to other entities (e.g. it may be the end of the current day, or the current year, in UTC time), but the revocation count may not. If a trusted and secure third party service such as DNSSEC is available for providing the PKG identity chain, it may also provide the base identity including the revocation count, upon being queried with an atomic identity.

V. Conclusion

We have described protocol design patterns that take advantage of ID-based encryption to implement secure channels without requiring additional roundtrips for key establishment or transmission of certificate chains, including patterns that provide responder-only and mutual authentication, and patterns that provide forward secrecy starting with the second flow. These patterns can be used in the design of new secure channel protocols, or can be retrofitted into existing protocols. They should be particularly useful for machine-to-machine communication in safety-critical and other applications that do not tolerate excessive latency. Practical feasibility will depend on identifying ID-based cryptosystems with good performance and low energy consumption. Performance benchmarks of the Pairing-Based Cryptographic Library [26] suggest that such cryptosystems can be found.

References

[1] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," August 2008, http://tools.ietf.org/html/rfc5246.

[2] S. Kent and K. Seo, "Security Architecture for the Internet Protocol," December 2005. http://tools.ietf.org/html/rfc4301.

[3] T. Ylonen, "The Secure Shell (SSH) Protocol Architecture," Jan. 2006, http://tools.ietf.org/html/rfc4251.

[4] C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)," September 2010. http://tools.ietf.org/html/rfc5996.

[5] Y. Sheffer and H. Tschofenig, "Internet Key Exchange Protocol Version 2 (IKEv2) Session Resumption," January 2010. http://tools.ietf.org/html/rfc5723.

[6] ZigBee Alliance, "ZigBee Specification FAQ," Question 8: What are the IEEE 802.15.4 technical attributes on which the ZigBee specification is based? Retrieved June 5, 2014 from http://www.zigbee.org/Specifications/ZigBee/FAQ.aspx.

[7] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State," RFC 5077. http://tools.ietf.org/html/rfc5077.

[8] A. Langley, "Message 06624 on the TLS Working Group Mailing List," http://www.ietf.org/mail-archive/web/tls/current/msg06624.html.

[9] H. Shacham, D. Boneh, and E. Rescorla, "Client-Side Caching for TLS," *ACM Transactions on Information and System Security*, vol. 7, no. 4, pp. 553–575, November 2004.

[10] A. Langley, N. Modadugu, and B. Moeller, "Transport Layer Security (TLS) False Start," draft-bmoeller-tls-falsestart-00, June 2, 2010. https://tools.ietf.org/html/draft-bmoeller-tls-falsestart-00.

[11] A. Langley, "False Start's Failure (11 Apr 2012)," https://www.imperialviolet.org/2012/04/11/falsestart.html.

[12] ——, "Transport Layer Security (TLS) Snap Start," May 2010. https://www.imperialviolet.org/binary/draft-agl-tls-snapstart-00.html.

[13] Internet Engineering Task Force, "Simple Public Key Infrastructure (spki)," Concluded WG. http://datatracker.ietf.org/wg/spki/.

[14] R. Moskowitz and P. Nikander, "Host Identity Protocol (HIP) Architecture," May 2006. http://tools.ietf.org/html/rfc4423.

[15] P. A. Lambert, "Key Centric Identity and Privilege Management," Presentation at the 2012 NIST Cryptographic Key Management Workshop. http://csrc.nist.gov/groups/ST/key_mgmt/documents/Sept2012_Presentations/LAMBERT_CKMW2012.pdf.

[16] C. Mulkey and D. C. Kar, "Identity-based encryption protocol for privacy and authentication in wireless networks," in *Network Security Technologies: Design and Applications*, A. Amine, O. A. Mohamed, and B. Benatallah, Eds. IGI Global, Nov. 2014, pp. 129–155.

[17] A. Shamir, "Identity-Based Cryptosystems and Signature Schemes," in *International Cryptology Conference*, 1984, pp. 47–53, http://link.springer.com/content/pdf/10.1007%2F3-540-39568-7_5.pdf.

[18] X. Boyen, "A tapestry of identity-based encryption: practical frameworks compared," *International Journal of Applied Cryptography*, vol. 1, pp. 3–21, 2008.

[19] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," May 2010, http://tools.ietf.org/html/rfc5869.

[20] K. Bhargavan, A. Delignat-Lavaud, C. Fournety, A. Pironti, and P.-Y. Strub, "Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS," March 2014. https://secure-resumption.com/tlsauth.pdf.

[21] The Chromium Projects, "QUIC, a multiplexed stream transport over UDP," http://www.chromium.org/quic.

[22] E. Barker, L. Chen, A. Roginsky, and M. Smid, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography," NIST SP 800-56A Rev. 2. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf.

[23] D. Boneh and X. Boyen, "Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles," in *EUROCRYPT*, 2004, pp. 223–238, http://crypto.stanford.edu/~xb//eurocrypt04b/bbibe.pdf.

[24] NIST, "Improving Trust in the Online Marketplace," April 10–11, 2013. http://www.nist.gov/itl/csd/ct/ca_workshop.cfm.

[25] P. Hoffman and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA," August 2012. http://tools.ietf.org/html/rfc6698.

[26] B. Lynn, "Pairing-Based Cryptographic Library Benchmarks," Retrieved on June 7, 2014 from http://crypto.stanford.edu/pbc/times.html.