Advanced Client/Server Authentication in TLS

Adam Hess, Jared Jacobson, Hyrum Mills, Ryan Wamsley, Kent E. Seamons, Bryan Smith
Computer Science Department
Brigham Young University
Provo, Utah, USA 84602
seamons@cs.byu.edu

ABSTRACT

Many business transactions on the Internet occur between strangers, that is, between entities with no prior relationship and no common security domain. Traditional security approaches based on identity or capabilities do not solve the problem of establishing trust between strangers. New approaches to trust establishment are required that are secure, scalable, and portable. One new approach to mutual trust establishment is trust negotiation, *the bilateral exchange of digital credentials to establish trust gradually. This paper describes the Trust Negotiation in TLS (TNT) protocol, an extension to the TLS handshake protocol that incorporates recent advances in trust negotiation into TLS to provide advanced client/server authentication in TLS. In this paper we describe the current limitations in TLS client/server authentication with respect to trust establishment, and show how the TNT protocol overcomes them. We also describe our implementation of TNT, built using PureTLS, a Java TLS package that is freely available. This implementation is the first to provide confidential trust negotiation, verification of private keys during trust negotiation, and a single trust negotiation protocol supporting interoperable trust negotiation strategies.*

1. Introduction

Many interactions on the Internet occur between strangers, that is, between entities with no prior relationship and no common security domain. Traditional security approaches based on identity or capabilities do not solve the problem of authentication between strangers, because strangers do not share a common security domain. New approaches are required that are secure, scalable, and portable.

When a server conducts a secure on-line transaction with a stranger, two common security problems must be addressed: first, the details of the transaction must remain confidential; second, the server must authenticate the client according to a pre-existing policy that specifies who is to be considered trustworthy for the given transaction. In our model, authentication of the client is not based on identity but rather on attribute values encoded in digital credentials—the online analogues of today's paper credentials.

Transport Layer Security (TLS) [4], the IETF adaptation of Netscape's SSL V3.0 [5] protocol, can provide confidentiality. TLS also provides client and server authentication. However, TLS authentication is not suitable for strangers trying to conduct business transactions. For example, a business may be interested in providing a service to residents of a given state. The identity of the client accessing the service is irrelevant to this decision; the client only needs to establish proof of residency. As discussed further in section 3, TLS does not provide this form of authentication.

Clients may have their own criteria for trusting servers. For example, before the client discloses a credential he or she considers to be sensitive, such as a credit card number and expiration date, the client may first require a credential from the server stating that the server will handle the client's private information properly. Without this assurance, customers who do not want their sensitive information disclosed to others will go elsewhere.

An interesting case to consider occurs when the server must authenticate the client in order to provide a service, but the client considers those credentials sensitive. The client therefore wants to authenticate the server before disclosing them. In such a case, a strictly one-way model of authentication will not suffice; mutual authentication is necessary.

Our approach to mutual trust establishment is called *trust negotiation*, the bilateral exchange of digital credentials to establish trust gradually [10][13][14][15]. Digital credentials contain digitally signed assertions by a credential issuer about a credential owner. A credential uses name/value pairs to describe one or more attributes of the owner. Each credential also contains the public key of the credential owner and is signed using the issuer's private key. The owner can answer challenges and otherwise demonstrate ownership of the credentials. Credentials are a more general name for certificates, such as X.509v3 certificates [8].

As an example of trust negotiation, an on-line bookstore may offer discounts to students at accredited universities. When a first-time customer requests a student discount, he or she will not have prior knowledge of the bookstore’s requirements for proof of student status. One approach is for the server to transmit a policy to the client. Such a policy could specify that the customer must submit a student ID and a credit card number in order to make an on-line purchase and receive a student discount. The customer (for example, a female student) is only willing to disclose her credit card number to a business that is a member of the Better Business Bureau (BBB). In accordance with her policy, her trust negotiation agent discloses her student ID and requests that the server return a BBB member credential to the client. The server then sends the client a BBB member credential. Finally, the client submits a valid digital credit card number and receives the student discount.

Since digital credentials can often contain sensitive information, associating an access control policy with each credential controls credential disclosure. As in the example above, a credential is disclosed only when its access control policy has been met. For a trust negotiation to be successful, some credentials must be freely available on at least one side of the negotiation.

This paper describes the Trust Negotiation in TLS (TNT) protocol, an extension to the TLS handshake protocol that incorporates trust negotiation to provide advanced client/server authentication in TLS. The TNT protocol is based on recent advances in trust negotiation and provides a solution for confidential trust negotiations and for verifying credential ownership. The details of these issues have so far not been addressed in past trust negotiation protocol proposals. This paper also describes our implementation of TNT, which extends PureTLS, a freely available Java implementation of SSL/TLS (See http://www.rtfm.com/puretls/ for more details). Section 2 discusses recent advances in the area of trust negotiation. Section 3 describes the TLS handshake protocol and identifies current limitations in TLS client/server authentication with regard to mutual authentication between strangers. Section 4 contains the description of the TNT protocol that extends the TLS handshake protocol. Section 5 describes an implementation of TNT, and Section 6 discusses related work. Section 7 contains conclusions and future work plans.

2. Trust negotiation

In our approach to automated trust establishment, trust is established incrementally by exchanging credentials and requests for credentials, an iterative process known as *trust negotiation* [10][13][14][15]. While a *trust negotiation protocol* defines the ordering of messages and the type of information messages will contain, a *trust negotiation strategy* controls the exact content of the messages, i.e., which credentials to disclose, when to disclose them, and when to terminate a negotiation.

Figure 1 introduces our TrustBuilder architecture for trust negotiation. A security agent mediates access to local protected *resources*: services, access control policies, and credentials. We say a credential or access control policy is *disclosed* if it has been sent to the other party in the negotiation, and that a service is disclosed if the other party is given access to it. Disclosure of protected resources is governed by access control policies.

The architecture in figure 1 supports a single protocol for establishing trust. The architecture is designed to support customized negotiation strategies. All trust negotiation strategies share the goal of building trust through an exchange of digital credentials. The purpose of this exchange is obtaining access to a protected resource. Once the access control policy for a particular credential has been satisfied, a local negotiation strategy must determine whether the credential is relevant to the current stage of the negotiation. If so, it will be disclosed. Different negotiation strategies will use different definitions of relevance, involving tradeoffs between computational costs, the length of the negotiation, and the number of disclosures.

From the handful of trust negotiation strategies proposed so far in the literature [10][13][14][15], it is clear that there are endless variations in how to negotiate trust. It is unlikely that a single strategy will meet the needs of all users. The TrustBuilder architecture is designed to support a strategy-independent, policy-language-independent trust negotiation protocol that ensures interoperability within a family of negotiation strategies [15].

Access control policies for local resources specify credentials that the other negotiation participant must disclose in order to obtain access to those resources. During a negotiation, the security agent invokes a local compliance checker in two ways. First, the security agent receives credentials from the other participant and checks to see if the relevant local access control policies are satisfied by the remote credentials before disclosing a local protected resource. Second, the agent may also receive

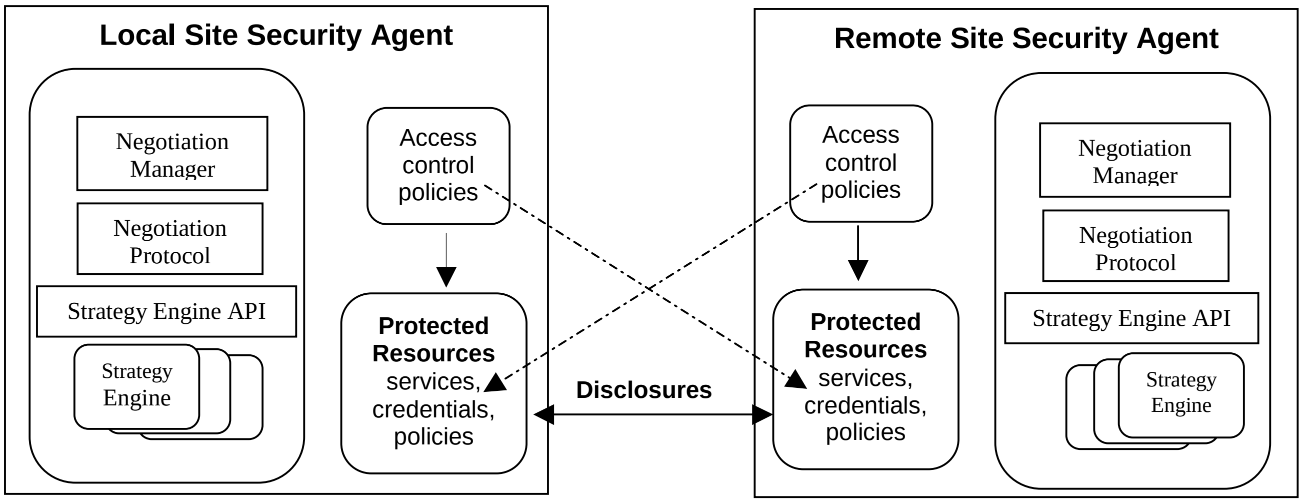

Figure 1. The TrustBuilder architecture for automated trust negotiation. A security agent who manages local protected resources and their associated access control policies represents each negotiation participant. TrustBuilder provides the necessary middleware support for security agents to enable negotiation strategy interoperability.

remote access control policies that serve as requests for local credentials. The agent checks to see whether any local credentials satisfy the remote access control policy. If so, the agent uses the negotiation strategy to determine if and when those local credentials should be disclosed to the other party to advance negotiation toward the goal of granting access to the protected resource.

When an access control policy ***P*** contains sensitive information, then ***P*** itself requires protection in the form of an access control policy for access to ***P***. Earlier work in trust negotiation introduced support for sensitive policies using policy graphs [10]. The presence of sensitive policies requires that trust be established gradually. For example, suppose a client begins an interaction with an unfamiliar web server. Before sending a sensitive request for credentials to the server that would reveal information regarding the nature of the client's business, the client may request credentials attesting to how the server handles private information and whether or not the server conforms to certified security practices. Once the client has established this initial level of trust, the client can continue by sending the sensitive request for further credentials from the server.

The previous work in trust negotiation has focused on support for sensitive credentials and access control policies, the definition and interoperability of trust negotiation strategies, and a trust negotiation protocol. In [15], a trust negotiation protocol was presented, along with the Disclosure Tree Strategy (DTS) family. It was proved that if two participants each choose a strategy from the DTS family, they will be able to negotiate trust just as well as if they had both adopted the same strategy. The issues of confidentiality and verifying ownership of submitted credentials during trust negotiation have not been addressed previously. To date, no implementation of the negotiation protocol or negotiation strategies exists.

3. Transport Layer Security (TLS)

TLS is a connection-oriented protocol that provides a secure channel between a client and a server. TLS supports confidentiality, data integrity, and client/server authentication. The TLS handshake protocol provides a means for authentication and the negotiation of security parameters, such as the encryption algorithms, encryption keys, MAC keys, etc., that are used to transmit data securely. The TLS record protocol specifies how application data is actually transmitted between two communicating hosts so that confidentiality and data integrity are provided.

The focus of the research described in this paper is authentication. Client/server authentication in TLS is handled in the handshake protocol. In this section, we describe the TLS handshake protocol for client/server authentication and identify the limitations in the protocol for authenticating strangers on the Internet.

The general TLS handshake protocol is illustrated in Figure 2, with optional messages shaded. The exact sequence of messages in a given handshake between a client and server will vary depending on the key exchange method selected by the client and server during the handshake. The TLS handshake has four phases. In the first phase, the client and server exchange hello messages that are used to establish security parameters used in the TLS session and settings used during the handshake, such as the key exchange algorithm. During the second phase,

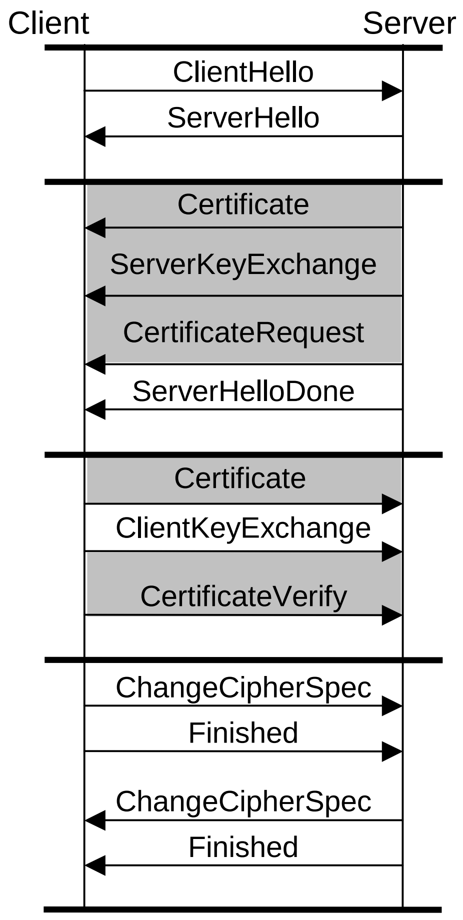

Figure 2. The general TLS handshake protocol with optional messages shaded.

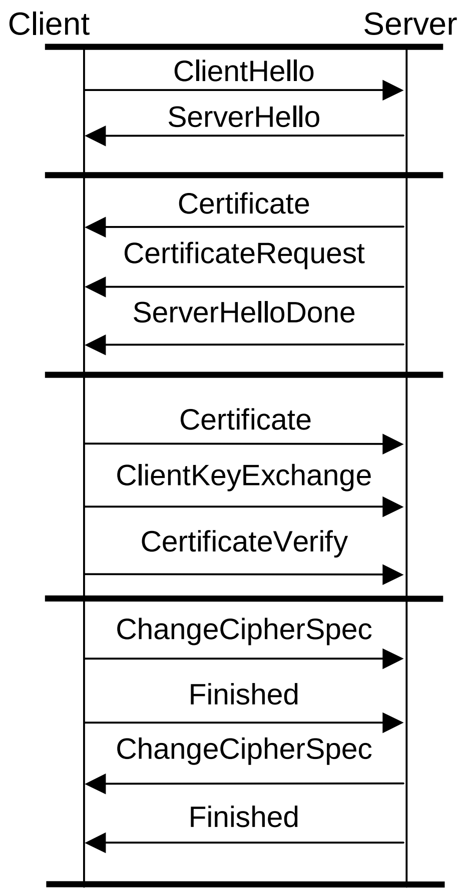

Figure 3. The TLS handshake protocol for client and server authentication using the RSA key exchange method.

the server sends a `Certificate` message to the client that may include a server certificate when an RSA key exchange is used, or Diffie-Hellman parameters when a Diffie-Hellman key exchange is used. The server may also request a certificate from the client using the `CertificateRequest` message. During the third phase of the handshake, the client may send its certificate to the server in a `Certificate` message along with a `CertificateVerify` message so that the server can verify certificate ownership, if the server requested a client certificate during the second phase. The client must send either a pre-master secret encrypted using the server's public key, or public Diffie-Hellman parameters, in the `ClientKeyExchange` message so that the client and server can compute a shared master secret. In the fourth phase of the handshake, the client and server finish the handshake so that they may begin exchanging application data.

The full range of handshake variants is beyond the scope of this paper. Interested readers are referred to [4][9][12] for a full treatment of the TLS handshake.

3.1. TLS client/server authentication

This section describes the specific form of a TLS handshake that supports client and server authentication using the RSA key exchange method, shown in Figure 3. Using this method, the client and server exchange certificates with one another for mutual authentication.

The client initiates the handshake by sending a `ClientHello` message to the server. The server responds with a `ServerHello` message. These messages contain the necessary information to establish the security parameters for the TLS session. Although the messages contain the same parameter types, they have a slightly different meaning. Table 1 lists the parameters of these two hello messages and describes their meaning.

The cipher suite parameter is a 4-tuple specifying the cryptographic algorithms to use in the TLS session. These include the server authentication algorithm, key exchange algorithm, bulk encryption algorithm, and digest algorithm for message integrity.

Parameters	ClientHello	ServerHello
Version	Highest SSL/TLS version supported by client	Lower of the client-suggested version and highest server-supported version
Random	Client-generated random structure, used as a nonce	Server-generated random structure
SessionID	Variable-length session identifier. A zero value indicates a new session. A non-zero value refers to an earlier session the client wishes to resume.	If client sends a zero value, server returns a new session ID, otherwise returns the old session ID supplied by the client.
CipherSuite	List of cryptographic algorithm combinations the client supports, in decreasing order of preference.	Single cipher suite selected from the list supplied by the client.
Compression Method	List of the compression methods supported by the client	Compression method selected by the server.

Table 1. A description of the parameters contained in the TLS ClientHello and ServerHello messages.

The server continues the handshake by sending a `Certificate` message containing an X.509 certificate or certificate chain. Next, the server sends a `CertificateRequest` message, communicating the following three items of information to the client: first, that the server requires a client certificate for authentication purposes; second, a list of certificate types the server is willing to accept; and third, a list of X.500 distinguished names of the certificate authorities that the server trusts. For the kind of handshake being described, the server specifies the certificate type as RSA. The list of trusted certificate authorities assists the client in selecting a certificate or certificate chain to submit that is signed by a root CA that the server trusts. Finally, the server sends a `ServerHelloDone` message indicating that it is now the client's turn to continue the handshake.

The third phase of the handshake protocol consists of messages sent from the client to the server. First, the client sends a `Certificate` message to the server containing an X.509 certificate or certificate chain.

Next, a `ClientKeyExchange` message is sent containing a client-generated, pre-master secret to be used for key generation. The client encrypts the pre-master secret using the public key contained in the server's certificate that was received earlier in the handshake. The only way for the server to decrypt the message successfully and obtain the pre-master secret is if the server possesses the private key associated with the certificate the server previously sent to the client. Thus, the `ClientKeyExchange` serves as an implicit challenge for the server to prove ownership of the private key. If the server successfully decrypts the pre-master secret, the server can generate the correct keys to be used during the TLS session.

The third phase of the handshake concludes when the client sends a `CertificateVerify` message to prove ownership of the private key associated with the certificate the client just disclosed to the server. The message consists of a signed hash of all previous messages exchanged during the handshake. The server decrypts the message using the public key contained in the client certificate and compares the result to a hash of all the previous messages exchanged during the handshake. This message serves as an implicit challenge for the client to prove ownership of the private key associated with the client certificate.

The handshake now enters the final phase. The client sends a `ChangeCipherSpec` message, indicating that the client will now begin encrypting its communications using the new cryptographic keys that were just computed. Then the client sends a `Finished` message to the server containing a hash of all the preceding messages exchanged during the handshake. The server follows suit by sending its own `ChangeCipherSpec` and `Finished` messages to the client. Upon completion of the handshake, application data begins to flow through the secure channel. Note that no application data, such as an HTTP request, flows from the client to the server until after the encrypted session is established.

3.2. Limitations in TLS authentication

The following are limitations to authenticating strangers on the Internet using TLS client/server authentication:

1. Certificates are exchanged in plain text during the initial TLS handshake. This does not present a danger that an eavesdropper can intercept the certificate and misuse it. An attacker who intercepts a certificate is unable to pose as the owner of the certificate without obtaining possession of the associated private key, assuming due diligence on the part of authentication services to challenge for possession of the associated private key appropriately whenever a certificate is presented. However, exchanging certificates in the clear does introduce privacy risks whenever certificate contents are sufficiently sensitive that disclosing the certificate to anyone unauthorized to receive it is undesirable.
2. The client and the server are limited to disclosing a single certificate chain to each other. In certain

circumstances, disclosing multiple certificate chains provides a greater level of trust compared to a single certificate chain, especially when several attributes are of interest in determining trust and different certification authorities are trusted to issue certificates containing those attributes. Requiring multiple certificate chains for authentication may also mean that multiple private keys must be compromised in order for an imposter to successfully breach the system.

3. The server specifies a list of distinguished names of certifying authorities that the server trusts when it requests a client certificate. In contrast, the client has no such opportunity.
4. The server discloses its certificate before the client discloses a certificate, forcing the server to always make the disclosure to a complete stranger.
5. The client always receives a certificate from the server before it is required to disclose its own certificate to the server. Although the client is able to verify the validity of the certificate contents, at the moment the client discloses a certificate to the server, the client has no proof that the server owns the certificate that it disclosed. Thus, the client cannot rely on the server certificate to determine conclusively whether or not to trust the server enough to disclose a sensitive client certificate.
6. If the certificate chain received by either the client or the server does not completely satisfy their authentication requirements, there is no facility in the protocol for requesting additional certificates to meet all the authentication requirements. The typical approach is for the client to authenticate the server if a trusted root CA known to the client signs the root certificate in the chain. Web clients typically authenticate the server if the distinguished name in the certificate matches the DNS name for the web server host machine. Most often, if servers authenticate clients at all, they simply verify that the client possesses the private key associated with the public key presented certificate.

4. Trust negotiation in TLS

The Trust Negotiation in TLS (TNT) protocol is an extension to the TLS handshake protocol that is designed to use trust negotiation to establish trust between strangers. TNT provides advanced client/server authentication in TLS by overcoming the limitations of TLS client/server authentication presented in the previous section. This section describes the TLS rehandshake and session resumption procedures and details how TNT leverages those procedures to support confidential trust negotiations. This section also describes the TNT protocol for conducting trust negotiation between a TLS client and server during a rehandshake. Finally, the definition of new messages added to TNT that are not already included in TLS is given using the syntax from the TLS protocol specification.

4.1. TLS rehandshake

Once a TLS connection is established using the handshake protocol described in the previous section, it is possible to conduct a TLS rehandshake. The rehandshake is simply the TLS handshake performed over an existing, and likely encrypted, TLS connection. (Although not common, it is possible for a TLS session to provide message integrity, but not encryption.) Either the client or server can initiate a rehandshake.

A client initiates a rehandshake by sending a new `ClientHello` message to the server after a previous handshake has finished. A server can initiate a rehandshake by sending a `ServerHelloRequest` message to a client. The client responds with a `ClientHello` message, and the handshake continues as usual. Either party has the option of ignoring a request for a rehandshake, but that is rarely done.

The three purposes of a rehandshake in SSL/TLS are: 1) client authentication, 2) cipher suite upgrading, and 3) replenishment of keying material. Suppose a server is configured to enforce different security requirements according to the specific data or service being accessed. Until the server receives the client’s request, there is no basis for client authentication. In this case, it is not possible to authenticate the client during the initial TLS handshake at the beginning of a connection, because the server has not yet received any application data from the client. Only after receiving application data, such as an HTTP request, can the server authenticate the client based on the access control policy of the requested service.

The rehandshake is used to upgrade the strength of the cipher suite in Netscape's Step-Up methodology—similar to Microsoft's Server Gated Cryptography, or SGC [9]. Before the United States’ export regulations were relaxed, an exception was granted for the use of strong encryption during certain financial transactions. For example, a client using only 40-bit encryption could automatically switch to stronger encryption when the client determined it had connected to a web server that was granted an exception to use strong cryptography, such as a financial institution outside the United States. For example, suppose a client connects to a bank in Paraguay and indicates that it is an exportable client who is only able to support 40-bit encryption. The bank's server can send back a special certificate in the TLS handshake indicating that it is allowed to use strong cryptography with exportable clients. After receiving this certificate, the client can initiate a rehandshake to negotiate a TLS session with stronger encryption.

Using Netscape's Step-Up methodology, a web browser initiates a re-handshake at the conclusion of the handshake. Microsoft's SGC methodology is implemented to allow the browser to interrupt the current handshake and begin a rehandshake. Although this is arguably more efficient, it does violate the SSL/TLS specification, which does not allow for a rehandshake to begin in the middle of a handshake.

A rehandshake is also used to replenish keying material. This is done to prevent cryptographic attacks on long-lived sessions. In TLS, the keying material can consist of up to six different values, depending on the cipher suite being used. This includes an encryption key, MAC key, and an initialization vector (IV) for both the client and server. Keys may also need to be replenished to prevent CBC rollover when large amounts of data are being transmitted using a block cipher in CBC mode. When two data blocks ***i*** and ***j*** encrypt to the same value ***c***, if blocks ***i+1*** and ***j+1*** are equal, they will also encrypt to the same value, revealing patterns in the data to a cryptanalyst.

Since trust negotiations involve sensitive certificates, negotiations must be confidential. During TLS client/server authentication, certificates are exchanged in plain text in an initial TLS handshake. To overcome this limitation, TNT is intended for use only during a rehandshake initiated during an encrypted TLS session, to keep the trust negotiation confidential.

This paper assumes the server initiates the rehandshake, to establish trust in a client according to the access control policy associated with the sensitive resource that the client has requested to access. In the future, TNT will also permit a client to establish trust in a server prior to any application data being passed through the secure channel, known as *client-initiated trust establishment* [1].

Single-round trust negotiations do not involve sensitive certificates, and therefore do not require confidentiality. Although not addressed in this paper, this simple case could be supported in the normal TLS handshake. A simple negotiation occurs, for example, when a server requests a certificate from the client and the client immediately discloses the certificate. TNT is designed to protect sensitive certificates in more complex negotiations.

4.2. TLS session resumption

The performance bottleneck in TLS handshakes is the public key cryptographic operations [9]. In particular, the encryption and decryption required to confidentially exchange a pre-master secret is expensive. One of the reasons the client must verify the server's certificate is to use the server's public key to encrypt the pre-master secret in the key exchange.

TLS supports session resumption in order to avoid the overhead of a full TLS handshake. With session resumption, an abbreviated handshake occurs as follows. The client sends a `ClientHello` message to a server and includes the `sessionID` from a previous session with the server. If the server is willing to resume the session, the server replies by returning the same `sessionID` in the `ServerHello` message. In order to resume a session, the client and server reuse the master secret from the prior session to compute new keying material, thus avoiding the expensive public-key operations of a normal handshake. After the `ServerHello` message they simply exchange `ChangeCipherSpec` and `Finished` messages, with the server proceeding first.

The TNT protocol presented in the next section leverages TLS session resumption in order to avoid the overhead of needlessly generating a new master secret. Once the client and server successfully negotiate trust, an abbreviated handshake takes place, similar to session resumption. Instead of completing the full handshake, the client and server compute new keying material by reusing the master secret from the current TLS session and conclude by sending `ChangeCipherSpec` and `Finished` messages to one another, with the server proceeding first.

4.3. TNT protocol

The TNT protocol, shown in figure 4, is designed to support trust negotiation between a TLS client and server whenever a TLS client has requested access to a sensitive service and the access control policy associated with the service is not satisfied. A TLS server desiring to negotiate trust with a client initiates a rehandshake by sending a `HelloNegotiationRequest` message to the client. The client responds with a `ClientHello` message, followed by the server sending a `ServerHello` message. The `sessionID` included in the hello messages is the `sessionID` associated with the current session, to allow for streamlined session resumption following a successful trust negotiation.

In order to conduct a successful trust negotiation, the negotiation participants must first agree on a trust negotiation strategy family. Once they agree on a strategy family, each party is free to independently select a negotiation strategy from that strategy family, with the guarantee that trust will be successfully negotiated whenever possible [15]. Two design choices for adding information on the negotiation strategy family into TLS are: 1) include the strategy family in the hello messages, or 2) incorporate the strategy family into the TLS cipher suite. A current IETF Internet draft specifies extensions to the `ClientHello` and `ServerHello` messages that can be used to communicate support for new capabilities

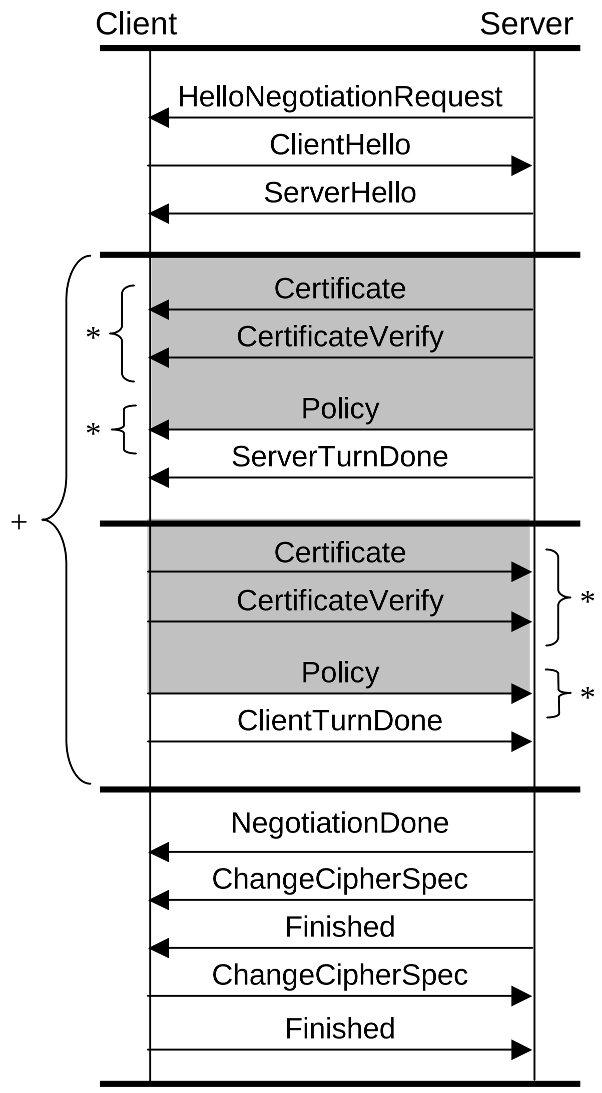

Figure 4. The TNT handshake protocol for negotiating trust during an encrypted TLS rehandshake, with optional messages shaded.

in a TLS client or server [2]. The draft is work in progress.

Anticipating that extensibility in the hello messages will be available in TLS in the future, TNT extends the `ClientHello` and `ServerHello` messages to include the `TrustNegotiationStrategyFamily` field. Using that field in the `ClientHello` message, a TNT client includes a list of the negotiation strategy families that the client supports. The server selects a strategy family from the list presented by the client and returns it in the `ServerHello` message.

Following the exchange of hello messages, the TNT protocol enters the negotiation phase, in which the server and client take turns disclosing policies and certificates until the negotiation succeeds or is terminated. During the server's turn, the server first discloses zero or more certificates using a combination of TLS `Certificate` and `CertificateVerify` messages. Immediately following each `Certificate` message, the server demonstrates ownership of a private key using the `CertificateVerify` message, unless the server has previously demonstrated possession of the private key earlier in the negotiation. The `CertificateVerify` message is encrypted using the server's private key associated with the certificate the server just disclosed. It is sent to prove to the client that the server is the owner of the certificate. Next, the server discloses zero or more `Policy` messages. Policies are disclosed to provide hints to the client regarding the certificates the client can disclose to advance the negotiation. Finally, the server sends a `ServerTurnDone` message indicating that the server has nothing further to disclose during this round of the negotiation.

Next, the client takes a turn disclosing information to advance the negotiation, following the same general pattern as the server. The client first discloses zero or more certificates using a combination of `Certificate` and `CertificateVerify` messages. Immediately following each `Certificate` message, the client demonstrates ownership of a private key using the `CertificateVerify` message, unless the client has previously demonstrated possession of the private key earlier in the negotiation. Next, the client discloses zero or more policies to provide hints to the server regarding the certificates the server can disclose to advance the negotiation. The client concludes its turn during the current round of negotiation by sending a `ClientTurnDone` message.

The negotiation continues for a number of rounds until the server's policy governing the resource is satisfied or the negotiation is terminated. The server checks to see whether the policy is satisfied after each round of the negotiation. Once the policy is satisfied, the server successfully concludes the negotiation by sending the `NegotiationDone` message.

Either party may terminate the negotiation at any time using the facilities for terminating any other TLS handshake. The decision to terminate is dependent on the negotiation strategy. Previous work on trust negotiation characterizes important properties of negotiation strategies, including termination [10][13][14][15]. A production implementation of TNT could use these previous results to place practical limits on the number of rounds of negotiation to insure a negotiation does not continue indefinitely.

The final phase of the TNT protocol is very similar to the conclusion of the abbreviated TLS handshake. The server sends a `ChangeCipherSpec` message and a

`Finished` message. Once the client replies with the same two types of messages, the rehandshake is complete.

The following is an example trust negotiation between a client and server using TNT. Suppose a web browser and server support TNT. The user is a student purchasing a book from an online bookstore that offers discounts to students at accredited universities. Suppose the user requests a student discount while purchasing a book in the context of a TLS session. For this example, assume the client and server are initially unfamiliar with one another.

The server initiates a TLS rehandshake in order to authenticate the client as a valid student by sending a `HelloNegotiationRequest` message to the client. The client responds with a `ClientHello` message, followed by a `ServerHello` message from the server. These messages permit the client and server to select a trust negotiation strategy family and the RSA key exchange method.

Next, the server sends a `Policy` message to the client that describes the certificates the client must submit in order to receive the discount service. In this case, the server requires a valid student ID certificate and a credit card certificate. For this example, we ignore the details of the valid certificate chains that are required in practice. The server then sends a `ServerTurnDone message`. Suppose the client has the necessary certificates to obtain a discount, but considers them to be sensitive. In order to establish trust in the server, the client sends a `Policy` message requesting certificates from the server that demonstrate the server is a member of the Better Business Bureau and is certified by TRUSTe to follow its privacy practices to not share private information with any outside party. The client then sends a `ClientTurnDone` message, indicating it is done with this stage of the negotiation.

Next, assume the server possesses the certificates that satisfy the client's request. The server sends a `Certificate` message and a `CertificateVerify` message for each of the certificate chains associated with its BBB and TRUSTe certificates, followed by a `ServerTurnDone` message. Now that the server is authorized to receive the client's certificates, the client continues the negotiation by sending a `Certificate` message and a `CertificateVerify` message for the two certificates requested earlier by the server, the student ID and credit card. The client then sends a `ClientTurnDone` message.

Once the client has satisfied the server's requirements for the discount service, the server sends a `NegotiationDone` message. Finally, the handshake completes according to the abbreviated approach used during session resumption, with the server and client sending a `ChangeCipherSpec` message and a `Finished` message to one another.

4.4. Overcoming TLS authentication limitations

The TNT protocol overcomes the limitations of TLS client/server authentication described in section 3.2, as described in the following list corresponding to each of the limitations given previously.

1. The TNT protocol is conducted within the scope of a rehandshake during an encrypted TLS session. Sensitive certificates are not exchanged in plain text.
2. The client and the server can exchange multiple certificates during each round of a negotiation. Requiring multiple certificates from different certifying authorities can reduce the risk associated with a single private key compromise. This is the electronic equivalent of requiring multiple forms of ID.
3. The TNT protocol allows the server to go first. However, the server is not obligated to disclose all of its certificates before the client discloses any. The server can save more sensitive certificates for disclosure during later rounds of the negotiation. Another approach that could be explored is to have the participants negotiate who goes first. The decision of who proceeds first is related to the negotiation strategy; so further work in this area is warranted.
4. The client and server have equal opportunity to disclose policies to one another to specify their trust requirements.
5. The client and server both send verification messages to one another after disclosing a certificate that they own. This verifies ownership of associated private keys. It is necessary to prove this immediately in TNT so that the certificate can be reliably used to gain access to sensitive certificates of a negotiation counterpart.
6. The TNT protocol permits clients and servers who are strangers to inform each other regarding their requirements for establishing trust through the use of Policy messages. Even a negotiation strategy that does not make use of `policy` messages could allow a client or server the ability to begin disclosing less sensitive certificates and only disclose more sensitive certificates when absolutely necessary. TNT is more flexible than the one-time disclosure currently available in TLS.

4.5. TNT message syntax

The majority of the message types in the TNT protocol are standard TLS message types. The semantics of these messages remain the same in TNT as they are in TLS. The TNT protocol introduces five new message types. The following syntax describes four new message types in TNT that are simply for control flow purposes.

```
Struct {} HelloNegotiationRequest
Struct {} ServerTurnDone
Struct {} ClientTurnDone
Struct {} NegotiationDone
```

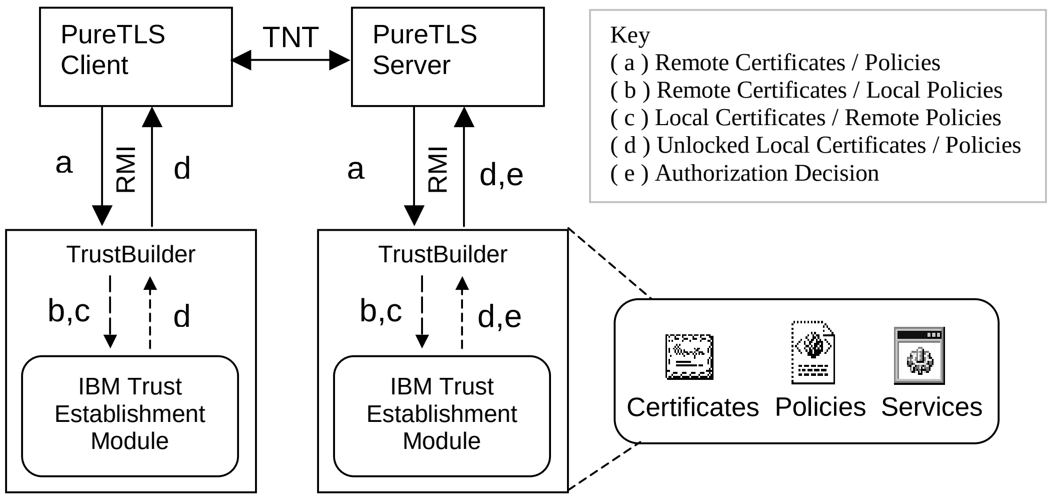
PureTLS Client
TNT
PureTLS Server
Key
(a) Remote Certificates / Policies
(b) Remote Certificates / Local Policies
(c) Local Certificates / Remote Policies
(d) Unlocked Local Certificates / Policies
(e) Authorization Decision
a
RMI
d
a
RMI
d,e
TrustBuilder
b,c
d
IBM Trust Establishment Module
TrustBuilder
b,c
d,e
IBM Trust Establishment Module
Certificates
Policies
Services

Figure 5. The implementation architecture for a trust negotiation prototype supporting the TNT protocol, an extension to the TLS handshake protocol. The implementation extends PureTLS, a freely available Java implementation of TLS. The architecture includes XML role-based access control policies and a compliance checker from the IBM Trust Establishment system.

The following syntax describes the TNT Policy message.

```
Struct {Policy policy;}
opaque Policy<2^24-1>
```

The syntax and semantics of the `Policy` message is not specified in the TNT protocol. It is dependent on the TNT implementation. Section 5 discusses the policy representation in our implementation of TNT.

5. Implementation

There are two approaches to supporting confidential trust negotiation using TLS: build an application level protocol above TLS, or integrate trust negotiation into TLS client/server authentication. The TNT protocol is an example of the latter approach. In our research, we are exploring both approaches through the design and development of TrustBuilder, trust negotiation middleware consisting of reusable trust negotiation components.

An advantage of integrating trust negotiation into TLS is the opportunity to leverage capabilities already available in a TLS implementation, including verifying certificate contents and both verifying and proving certificate ownership. An application-level protocol for trust negotiation requires a custom solution providing similar capabilities.

We have implemented a research prototype supporting the TNT protocol described earlier. The implementation architecture is illustrated in Figure 5. Our implementation extends PureTLS, a free Java-only implementation of the SSL V3.0 and TLS V1.0 protocols. PureTLS was developed by Eric Rescorla and is distributed by RFTM at http://www.rtfm.com/puretls/.

In our implementation, the client and server each rely on a TrustBuilder component to manage certificates, policies, and services. TrustBuilder implements the negotiation strategy and handles all decision-making aspects of a negotiation. Once the client and server receive remote certificates and policies during a negotiation, they submit them to TrustBuilder for evaluation. TrustBuilder determines which local certificates and policies to disclose, and when to disclose them.

In our research, we utilize the IBM Trust Establishment (TE) system [6] to create X.509v3 certificates and XML role-based access control policies that govern access to sensitive services and certificates. Additionally, the TE runtime system has a compliance checker that TrustBuilder uses to verify whether a set of certificates satisfies an access control policy. Note that the TLS client and server each rely on the compliance checker during trust negotiation to verify 1) whether the remote certificates received during the negotiation satisfy the local policies governing access to local services or certificates, and 2) whether local certificates satisfy remote policies received during the negotiation.

The typical usage model for a compliance checker in trust management systems is to input a set of certificates and a policy to a Boolean decision function. Our trust negotiation prototype requires an extended usage model whenever policies are disclosed during trust negotiation. For example, when a TLS client receives a policy from the TLS server, the client is able to search for local certificates that satisfy the server's policy, so it can submit those certificates to the server. This allows the negotiation to focus on only those certificates that can advance the negotiation to a successful conclusion. However, this requires that the compliance checker accept a set of local certificates and a remote policy and return not only a Boolean result indicating whether or not the policy is satisfied, but also the set of certificates that satisfy the policy so that they can be disclosed to the other party. The TE system developers provided an API to their compliance checker that supports the extended usage model, prompted by our requirements for trust negotiation.

In our implementation, the TrustBuilder trust negotiation manager and compliance checker run in a separate process, providing a flexible architecture for TLS clients and servers. For instance, multiple TLS servers operating in a high-performance web server environment can share the compliance checker, permitting all private key operations to be encapsulated within a separate process that can be isolated on a secure server with a cryptographic coprocessor.

Our experience demonstrates that the protocol extensions to TLS outlined in this paper can be readily incorporated into existing TLS implementations while still providing backward compatibility with existing TLS implementations. The TNT implementation provides a trust negotiation protocol supporting interoperable trust negotiation strategies [15] and provides the first implementation of confidential trust negotiation. A research prototype implementation of TNT is available from BYU's Internet Security Research Lab (http://isrl.cs.byu.edu/).

6. Related work

Yu et al. [15] introduce the notion of a family of trust negotiation strategies guaranteed to interoperate. They also introduce the idea of a trust negotiation protocol supporting a variety of negotiation strategies. Our work represents the design and implementation of those ideas in TNT, an extension of the TLS handshake protocol. Previous trust negotiation prototypes [13] [11] focused on languages for trust negotiation and negotiation strategies. The trust negotiation protocols were implemented at the application layer. Our work extends these earlier efforts by providing confidential trust negotiation and the verification of private keys associated with certificates disclosed during a trust negotiation. The TNT protocol is designed to support negotiation strategy interoperability.

Persiano et al. [7] introduce the SPSL protocol to extend TLS so that a portion of a disclosed certificate remains private from the party to which the certificate was disclosed. This is desirable when an attribute certificate contains some sensitive attributes that need not be disclosed in order to establish trust. Our work on trust negotiation focuses on establishing trust in another party in order to disclose a certificate to them. The two approaches complement one another and could be combined in order to authenticate another party prior to certificate disclosure as well as keep some private certificate contents completely confidential.

A recent IETF Internet draft from the TLS Working Group [2] discusses work in progress to define extensible hello messages in the TLS handshake protocol. Extensibility will allow a TLS client and server to negotiate additional features. One example taken from the draft document illustrating the use of extensions is to allow TLS clients to indicate to TLS servers which CA root keys they possess in the hello message. This example is one way to overcome a limitation to TLS authentication presented earlier in the paper in which a TLS client cannot inform a TLS server regarding the CAs they trust. The negotiation strategy family field added to the hello messages in TNT is another example of an item that could be included in an extensible hello message.

Dean et al. [3] describe an extension to TLS that uses client puzzles to prevent denial of service attacks on TLS servers. An imbalance in the computational requirements of TLS places an undue burden on the server. An attacker can exploit this to launch a successful denial of service attack. The use of client puzzles places additional computational demands on the client that tends to balance the load enough to discourage and prevent successful denial of service attacks. This paper introduces trust negotiation into TLS, potentially increasing the requirements on TLS servers. Adopting the use of client puzzles into our work has the potential to prevent denial of service attacks against TNT.

7. Conclusions and future work

This paper presents TNT, an extension to the TLS handshake protocol supporting advanced client/server authentication in TLS. TNT incorporates recent advances in trust negotiation from TrustBuilder [15] into TLS client/server authentication. This integration overcomes the existing limitations in TLS client/server authentication for establishing trust between strangers.

By integrating trust negotiation into TLS, the strengths of the TLS protocol are leveraged to provide confidential trust negotiation and the verification of private keys associated with certificates disclosed during a trust negotiation. TNT exploits the rehandshake facility of TLS to enable a TLS server to "pull" certificates from the client as needed, according to the access control policies of a sensitive resource accessed by a TLS client.

An implementation of TNT has been built that extends PureTLS, a Java TLS package that is freely available. This implementation is the first to provide confidential trust negotiation, the verification of private keys during trust negotiation, and a trust negotiation protocol designed to support interoperable trust negotiation strategies [15].

In the future, we will explore alternative trust negotiation strategies to insure that the current protocol is sufficiently general to support all useful negotiation strategy families. Also, policy languages for trust negotiation is an active area of research, and requirements for a trust negotiation policy language are emerging. We believe it unlikely that a single policy language will be universally adopted. We intend to examine current policy languages in terms of ease of use, expressiveness, and efficiency for use during trust negotiation. This exploration will help identify any needed extensions to TNT to support policy language interoperability, including additional negotiation parameters TLS handshake hello messages, such as certificate format and policy language.

Trust negotiation places greater computational demands on TLS servers, requiring further study of performance optimizations and scalable security architectures for TNT. Our research to date has not addressed performance. The current TNT prototype consists of non-optimized Java code.

At times, a client may want to authenticate a server prior to sending a sensitive service request, known as client-initiated trust establishment [1]. To illustrate when this might occur, suppose a client refuses to send personal information gathered by the service until the server discloses a TRUSTe certificate declaring that the server handles private information appropriately, such as not disclosing it to a third party unless the client provides explicit authorization. Since TLS allows a client to initiate a rehandshake, the client could establish trust in the server before any sensitive application data is transmitted, using a similar approach to the server-initiated rehandshake approach adopted by TNT in this paper.

This paper considered trust negotiations involving sensitive certificates, requiring that trust negotiations be confidential. This made it necessary that a trust negotiation always occur in the context of an encrypted TLS rehandshake. In the future, we will consider extending TLS to support the simplest negotiations not involving sensitive certificates. Simple negotiations could occur frequently in practice, for example, during client-initiated trust establishment when the client verifies that a TLS server satisfies a general, well-known security requirement that need not be kept confidential. These trust negotiations could be conducted during the initial TLS handshake, since they do not require confidentiality.

8. Acknowledgements

This research was supported by DARPA through AFRL contract number F33615-01-C-0336 and the Space and Naval Warfare Systems Center San Diego grant number N66001-01-18908. The authors express thanks to Marianne Winslett for her helpful feedback and discussions on an earlier version of this paper and to Ryan Jarvis for his help preparing the final submission. They also thank the anonymous reviewers for their comments.

9. References

[1] T. Barlow, A. Hess, and K. E. Seamons. Trust Negotiation in Electronic Markets. *Eighth Research Symposium on Emerging Electronic Markets*, Maastricht, Netherlands, September 2001.

[2] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. TLS Extensions. Internet Draft, TLS Working Group, June 20, 2001. Work in progress available at http://www.ietf.org/internet-drafts/draft-ietf-tls-extensions-00.txt.

[3] D. Dean and A. Stubblefield. Using Client Puzzles to Protect TLS. *Proceedings of the 10th USENIX Security Symposium*, Washington D.C., August 2001.

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

[5] A. Frier, P. Karlton, and P. C. Kocher. The SSL 3.0 Protocol. Netscape Communications Corp., Nov. 18, 1996.

[6] A. Herzberg, J. Mihaeli, Y. Mass, D. Naor, and Y. Ravid. Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. *IEEE Symposium on Security and Privacy*, Oakland, May 2000.

[7] P. Persiano and I Visconti. User Privacy Issues Regarding Certificates and the TLS Protocol. *7th ACM Conference of Computer and Communications Security*, Athens, Greece, November 2000.

[8] Recommendation X.509--Information Technology—Open Systems Interconnection--The Directory: Authentication Framework. International Telecommunication Union, Aug. 1997.

[9] E. Rescorla. SSL and TLS: Designing and Building Secure Systems. Addison-Wesley, 2001.

[10] K. E. Seamons, M. Winslett, and T. Yu. Limiting the Disclosure of Access Control Policies During Automated Trust Negotiation. *Symposium on Network and Distributed System Security*, San Diego, Feb. 2001.

[11] K. E. Seamons, W. Winsborough, and M. Winslett. Internet Credential Acceptance Policies. Proceedings of the Workshop on Logic Programming for Internet Applications, Leuven, Belgium, July 1997

[12] S. Thomas. SSL and TLS essentials: Securing the Web. Wiley Computer Publishing, 2000.

[13] W. Winsborough, K. E. Seamons, and V. Jones. Automated Trust Negotiation. *DARPA Information Survivability Conference and Exposition*, Hilton Head, SC, Jan. 2000.

[14] T. Yu, X. Ma, and M. Winslett. PRUNES: An Efficient and Complete Strategy for Automated Trust Negotiation over the Internet. *7th ACM Conference on Computer and Communications Security*, Athens, Greece, November 2000.

[15] T. Yu, M. Winslett, and K. E. Seamons. Interoperable Strategies in Automated Trust Negotiation. *8th ACM Conference on Computer and Communications Security*, Philadelphia, Pennsylvania, November 2001.