Google Cloud

Application Layer Transport Security

Cesar Ghali, Adam Stubblefield, Ed Knapp, Jiangtao Li, Benedikt Schmidt, Julien Boeuf

Table of Contents

Executive summary

- Google's Application Layer Transport Security (ALTS) is a mutual authentication and transport encryption system developed by Google and typically used for securing Remote Procedure Call (RPC) communications within Google's infrastructure. ALTS is similar in concept to mutually authenticated TLS but has been designed and optimized to meet the needs of Google's datacenter environments.

- The ALTS trust model has been tailored for cloud-like containerized applications. Identities are bound to entities instead of to a specific server name or host. This trust model facilitates seamless microservice replication, load balancing, and rescheduling across hosts.

- ALTS relies on two protocols: the Handshake protocol (with session resumption) and the Record protocol. These protocols govern how sessions are established, authenticated, encrypted, and resumed.

- ALTS is a custom transport layer security solution that we use at Google. We have tailored ALTS to our production environment, so there are some tradeoffs between ALTS and the industry standard, TLS. Section 5 discusses these tradeoffs in more detail.

1. Introduction

Production systems at Google consist of a constellation of microservices[1] that collectively issue O(10^{10}) Remote Procedure Calls (RPCs) per second. When a Google engineer schedules a production workload[2], any RPCs issued or received by that workload are protected with ALTS by default. This automatic, zero-configuration protection is provided by Google's Application Layer Transport Security (ALTS). In addition to the automatic protections conferred on RPC's, ALTS also facilitates easy service replication, load balancing, and rescheduling across production machines. This paper describes ALTS and explores its deployment over Google's production infrastructure.

Audience: This document is aimed at infrastructure security professionals who are curious about how authentication and transport security are performed at scale in Google.

Prerequisites: In addition to this introduction, we assume a basic understanding of cluster management at Google.

When a Google engineer schedules a production workload, any RPCs that workload issues has automatic, zero-configuration protection using Google's Application Layer Transport Security (ALTS).

2. Application-Level Security and ALTS

Many applications, from web browsers to VPNs, rely on secure communication protocols, such as TLS (Transport Layer Security) and IPSec, to protect data in transit[3]. At Google, we use ALTS, a mutual authentication and transport encryption system that runs at the application layer, to protect RPC communications. Using application-level security allows applications to have authenticated remote peer identity, which can be used to implement fine-grained authorization policies.

2.1 Why Not TLS?

It may seem unusual for Google to use a custom security solution such as ALTS when the majority of Internet traffic today is encrypted using TLS. ALTS began development at Google in 2007. At the time, TLS was bundled with support for many legacy protocols that did not satisfy our minimum security standards. We could have designed our security solution by adopting the TLS components we needed and implementing the ones we wanted; however, the advantages of building a more Google-suited system from scratch outweighed the benefits of patching an existing system. In addition, ALTS is more appropriate for our needs,

[1] A **microservice** is an architectural style that structures an application as a collection of loosely coupled services which implement business capabilities.
[2] A **production workload** is an application that Google engineers schedule to run in Google's datacenters.
[3] For more information on how Google protects data in transit, see our whitepaper, "Encryption in Transit in Google Cloud".

and historically more secure than older TLS. Listed below are the key differences between TLS and ALTS.

- There is a significant difference between the trust models[4] of TLS with HTTPS semantics and ALTS. In the former, server identities are bound to a specific name and corresponding naming scheme. In ALTS, the same identity can be used with multiple naming schemes. This level of indirection provides more flexibility and greatly simplifies the process of microservice replication, load balancing, and rescheduling between hosts.

- Compared to TLS, ALTS is simpler in its design and implementation. As a result, it is easier to monitor for bugs and security vulnerabilities using manual inspection of source code or extensive fuzzing.

- ALTS uses Protocol Buffer to serialize its certificates and protocol messages, while TLS uses X.509 certificates encoded with ASN.1. The majority of our production services use protocol buffers for communication (and sometimes storage), making ALTS a better fit for Google's environment.

ALTS is designed to be a highly reliable, trusted system that allows for service-to-service authentication and security with minimal user involvement.

2.2 ALTS Design

ALTS is designed to be a highly reliable, trusted system that allows for service-to-service authentication and security with *minimal* user involvement. To achieve this, the properties listed below are part of ALTS's design:

- **Transparency:** ALTS configuration is transparent to the application layer. By default, service RPCs are secured using ALTS. This allows application developers to focus on the functional logic of their services without having to worry about credential management or security configurations. During service-to-service connection establishment, ALTS provides applications with an authenticated remote peer identity which can be used for fine-grained authorization checks and auditing.

- **State-of-the-art cryptography:** All cryptographic primitives and protocols used by ALTS are up-to-date with current known attacks. ALTS runs on Google-controlled machines, meaning that all supported cryptographic protocols can be easily upgraded and quickly deployed.

- **Identity model:** ALTS performs authentication primarily by identity rather than host name. At Google, every network entity (e.g. a corporate user, a physical machine, or a production service or workload) has an associated identity. All communications between services are mutually authenticated.

[4] A trust model is the mechanism through with a security protocol identifies, distributes and rotates credentials and identities.

ALTS performs authentication primarily by identity rather than host. At Google, every network entity (e.g., a corporate user, a physical machine, or a production service) has an associated identity.

- **Key distribution:** ALTS relies on each workload having an identity, which is expressed as a set of credentials. These credentials are deployed in each workload during initialization, without user involvement. In parallel, a root of trust and a trust chain for these credentials are established for machines and workloads. The system allows for automatic certificate rotation and revocation without application developers involvement.
- **Scalability:** ALTS is designed to be very scalable in order to support the massive scale of Google's infrastructure. This requirement resulted in the development of efficient session resumption, see Section 4.3.
- **Long-lived connections:** Authenticated key exchange cryptographic operations are computationally expensive. To accommodate the scale of Google's infrastructure, after an initial ALTS handshake, connections can be persisted for a longer time to improve overall system performance.
- **Simplicity:** TLS by default comes with support for legacy protocol versions and backwards compatibility. ALTS is considerably simpler as Google controls both clients and servers, which we designed to natively support ALTS.

3. ALTS Trust Model

ALTS performs authentication primarily by identity rather than host. At Google, every network entity (e.g., a corporate user, a physical machine, or a production service) has an associated identity. These identities are embedded in ALTS certificates and used for peer authentication during secure connection establishment. The model we pursue is that our production services run as production entities that can be managed by our Site Reliability Engineers (SREs)[5]. The development versions of these production services run as test entities that can be managed by both SREs and developers.

For example, let's assume we have a product with two services: **`service-frontend`** and **`service-backend`**. SREs can launch the production version of these services: **`service-frontend-prod`** and **`service-backend-prod`**. Developers can build and launch development versions of these services, **`service-frontend-dev`** and **`service-backend-dev`**, for testing purposes. The authorization policy in the production services will be configured not to trust the development versions of the services.

3.1 ALTS Credentials

There are three types of ALTS credential, all of which are expressed in Protocol Buffer message format.

[5] Some services are managed directly by developers.

- **Master certificate:** signed by a remote Signing Service and used to verify handshake certificates. The master certificate contains a public key associated with a master private key, e.g., RSA keypair. This private key is used to sign handshake certificates. These certificates, when exercised in combination with the ALTS policy discussed below, are essentially constrained intermediate Certificate Authority (CA) certificates. Master certificates are typically issued for production machines and schedulers of containerized workloads such as the Borgmaster[6].

- **Handshake certificate:** created and signed locally by the master private key. This certificate contains the parameters used during the ALTS handshake (secure connection establishment), for example, static Diffie-Hellman (DH) parameters and the handshake ciphers. Also, the handshake certificate contains the master certificate that it is derived from, i.e., the one associated with the master private key that signs the handshake certificate.

- **Resumption key:** is a secret that is used to encrypt resumption tickets. This key is identified by a Resumption Identifier ID_R that is unique for, and shared among, all production workloads running with the same identity and in the same datacenter cell. For more details on session resumption in ALTS, see Section 4.3.

Figure 1 shows the ALTS certificate chain, which consists of a Signing Service verification key, a master certificate and a handshake certificate. The Signing Service verification keys are the root of trust in ALTS and are installed on all Google machines in our production and corporate networks.

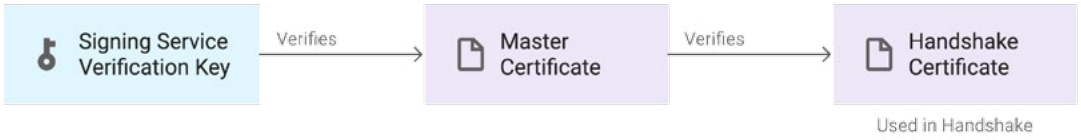


Figure 1: ALTS certificate chain

In ALTS, a Signing Service certifies Master certificates which in turn certify Handshake certificates. As Handshake certificates are created more often than Master certificates, this architecture reduces the load on the Signing service. Certificate rotation happens frequently at Google, especially for handshake certificates[7]. This frequent rotation compensates for the static key exchange pairs carried by the handshake certificates[8].

3.1.1 Certificate Issuance

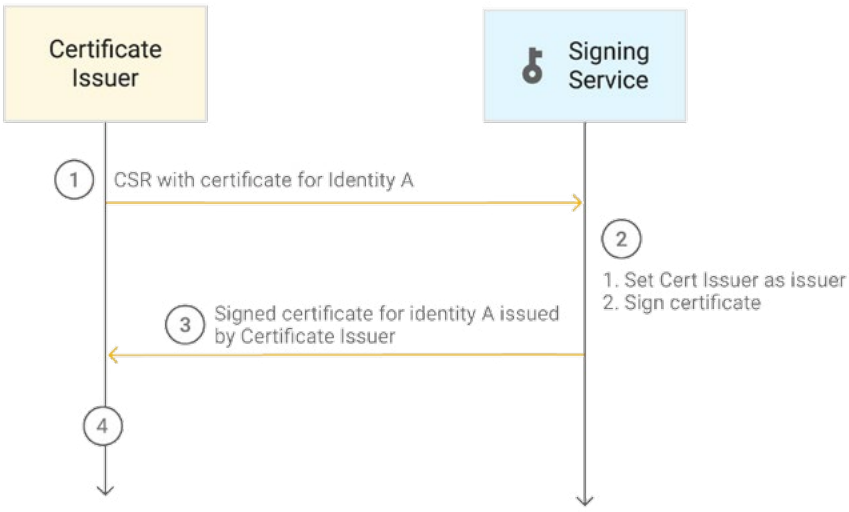
In order to participate in an ALTS secure handshake, entities on the network need to be provisioned with handshake certificates. First, the issuer obtains a master certificate signed by the Signing Service and optionally passes it down to the entity. Then, a handshake certificate is created and signed by the associated master private key.

[6] Borgmaster is responsible for scheduling and initializing Google production workloads. For more information see Large-scale cluster management at Google with Borg.
[7] More information about certificate rotation frequencies can be found in "Encryption in Transit in Google Cloud".
[8] If a key is compromised, only the traffic for the lifetime of this keypair will be discoverable by the attacker.

Typically, the issuer is our internal Certificate Authority (CA) when issuing certificates to machines and humans, or the Borgmaster when issuing certificates to workloads. However, it can be any other entity, e.g., a restricted Borgmaster for a test datacenter cell.

Figure 2 shows how the Signing service is used to create a master certificate. The process consists of the following steps.

Figure 2: Certificate Issuance



1. The Certificate Issuer sends a Certificate Signing Request (CSR) to the Signing Service. This request asks the Signing Service to create a certificate for identity A. This identity, for example, can be a corporate user or the identity of a Google production service.
2. The Signing Service sets the issuer of the certificate (included in the CSR) to the requester (the Certificate Issuer in this case) and signs it. Recall that the corresponding Signing Service public (verifying) key is installed on all Google machines.
3. The Signing Service sends the signed certificate back.
4. A handshake certificate is created for identity A and is signed by the master certificate associated private key.

As shown in the process above, with ALTS, the issuer and signer of a certificate are two different logical entities. In this case, the issuer is the Certificate Issuer entity while the signer is the Signing Service.

There are three common categories of certificates in ALTS, namely: Human, Machine, and Workload. The following sections outline how each of these certificates are created and used in ALTS.

3.1.2 Human Certificates

At Google, we use ALTS to secure RPCs issued by human users to production services. To issue an RPC, a user must provide a valid handshake certificate. For example, if Alice wants to use an application to issue an ALTS-secure RPC,

she can authenticate to our internal CA. Alice authenticates to the CA using her username, password, and two-factor authentication. This operation results in Alice getting a handshake certificate that is valid for 20 hours.

3.1.3 Machine Certificates

Every production machine in Google's datacenters has a machine master certificate. This certificate is used to create handshake certificates for core applications on that machine, e.g. machine management daemons. The primary identity embedded in a machine certificate refers to the typical purpose of the machine. For example, machines used to run different kinds of production and development workloads can have different identities. The master certificates are only usable by machines running verified software stacks; in some cases this trust is rooted in custom security hardware[9]. All production machine master certificates are issued by the CA and rotated every few months. Also, all handshake certificates are rotated every few hours.

3.1.4 Workload Certificates

A key advantage of ALTS is that it operates on the idea of a workload identity which facilitates easy service replication, load balancing, and rescheduling across machines. In our production network, we use a system called Borg[10] for cluster management and machine resource allocation at scale. The way that Borg issues certificates is part of the ALTS machine-independent workload identity implementation. The remainder of this section provides an overview of our workload certification.

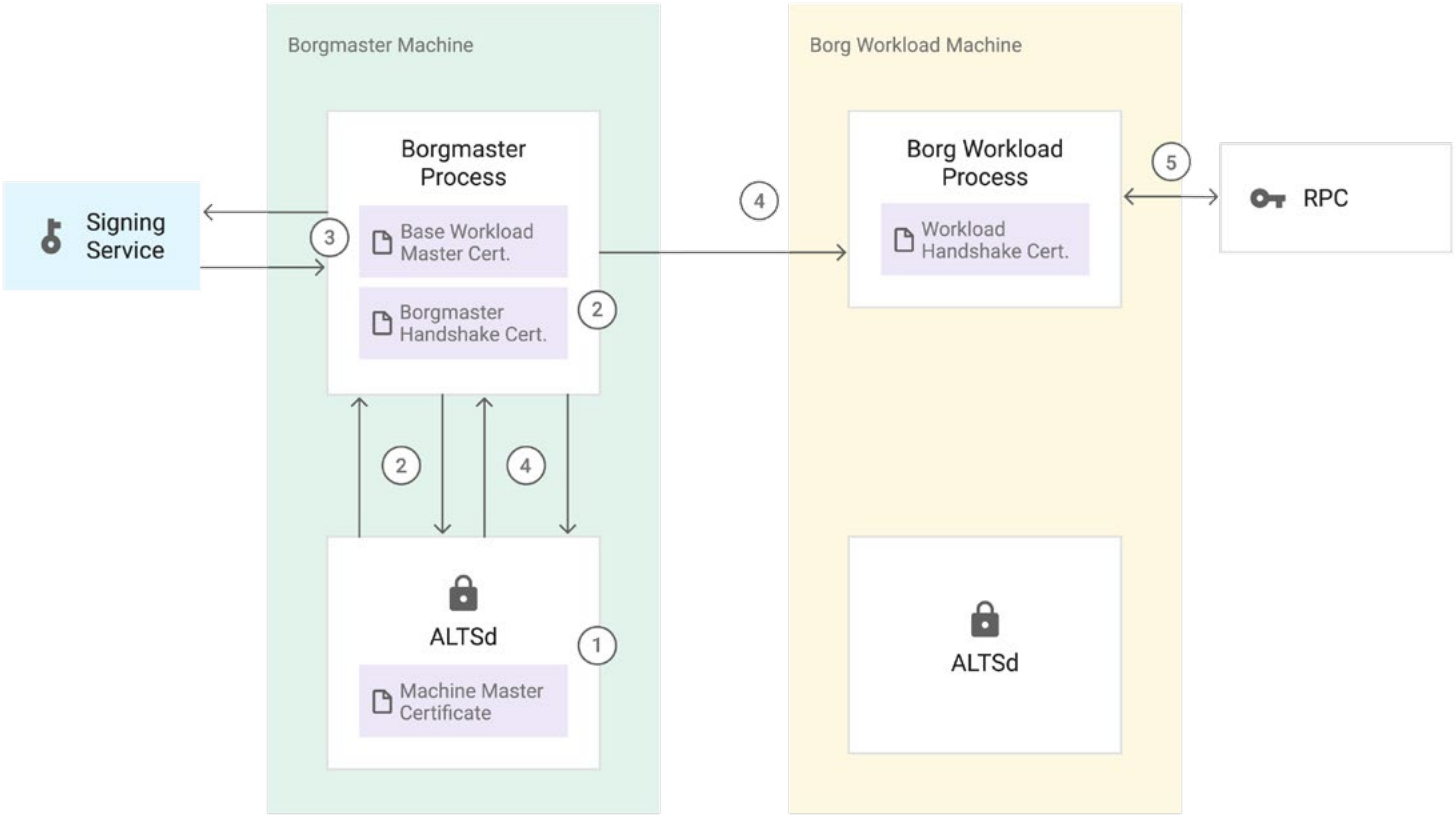
Each workload in our production network runs in a Borg cell. Each cell contains a logically centralized controller called the Borgmaster, and several agent processes called Borglets that run on each machine in that cell. Workloads are initialized with associated Workload Handshake Certificates issued by the Borgmaster. Figure 3 shows the process of workload certification in ALTS with Borg.

1. Each Borgmaster comes pre-installed with a Machine Master Certificate and associated private key (not shown in the diagram).
2. The ALTSd[11] generates a Borgmaster Handshake Certificate and signs it using the Machine Master private key. This Handshake Certificate allows Borgmaster to issue ALTS-secure RPCs.
3. The Borgmaster creates a Base Workload Master Certificate, and the corresponding private key. The Borgmaster initiates a request to get its Base Workload Master Certificate signed by the Signing Service. As a result, the Signing Service lists the Borgmaster as the issuer on this certificate.

The Borgmaster is now ready to schedule workloads that need to use ALTS. The steps below happen when a client schedules a workload to run on Borg as a given identity.

[9] Titan in depth: Security in plaintext.
[10] Large-scale cluster management at Google with Borg.
[11] ALTSd: a daemon responsible for, amongst other ALTS operations, the creation of handshake certificates.

Figure 3: Handshake Certificate Creation in the Google Production Network



4. The Borgmaster verifies that the client is authorized to run workloads as the identity that is specified in the workload configuration. If so, the Borgmaster schedules the Borg workload on the Borglet, and issues a Workload Handshake Certificate and its corresponding private key. This certificate is chained from the Base Workload Master Certificate. The Workload Handshake Certificate and its private key are then securely delivered to the Borglet (over a mutually authenticated ALTS protected channel between the Borgmaster and the Borglet). The Borgmaster rotates its Base Workload Master Certificate and reissues Handshake Certificates for all running workloads approximately every two days. In addition, each workload running as the same user in the same cell receives the same resumption key and identifier (ID_R) provisioned by the Borgmaster.
5. When the workload needs to make an ALTS-secure RPC, it uses the Workload Handshake Certificate in the handshake protocol. ID_R is also used as part of the handshake to initiate session resumption. For more information about session resumption in ALTS, see Section 4.3.

3.2 ALTS Policy Enforcement

The ALTS policy is a document that lists which issuers are authorized to issue certain categories of certificates for which identities. It is distributed to every machine on our production network. For example, the ALTS policy allows the CA to issue certificate to machines and humans. It also allows Borgmaster to issue certificates to workloads.

We have found that policy enforcement during certificate verification, as opposed to certificate issuance, is a more flexible approach as it allows for different policies to be enforced on different types of deployments. For example, we may want a policy in a test cluster to be more permissive than one in a production cluster.

During the ALTS handshake, the certificate validation includes a check of the ALTS policy. The policy ensures that the issuer listed in the certificate being validated is authorized to issue that certificate. If that is not the case, the certificate is rejected and the handshake process fails. Figure 4 illustrates how the policy enforcement works in ALTS. Following the scenario in Figure 2, assume that Mallory (a corporate user who wants to escalate her privileges) wants to issue a master certificate to the Network Admin, which is a powerful identity that can reconfigure the network. It goes without saying that Mallory is not authorized on the ALTS policy to perform this operation.

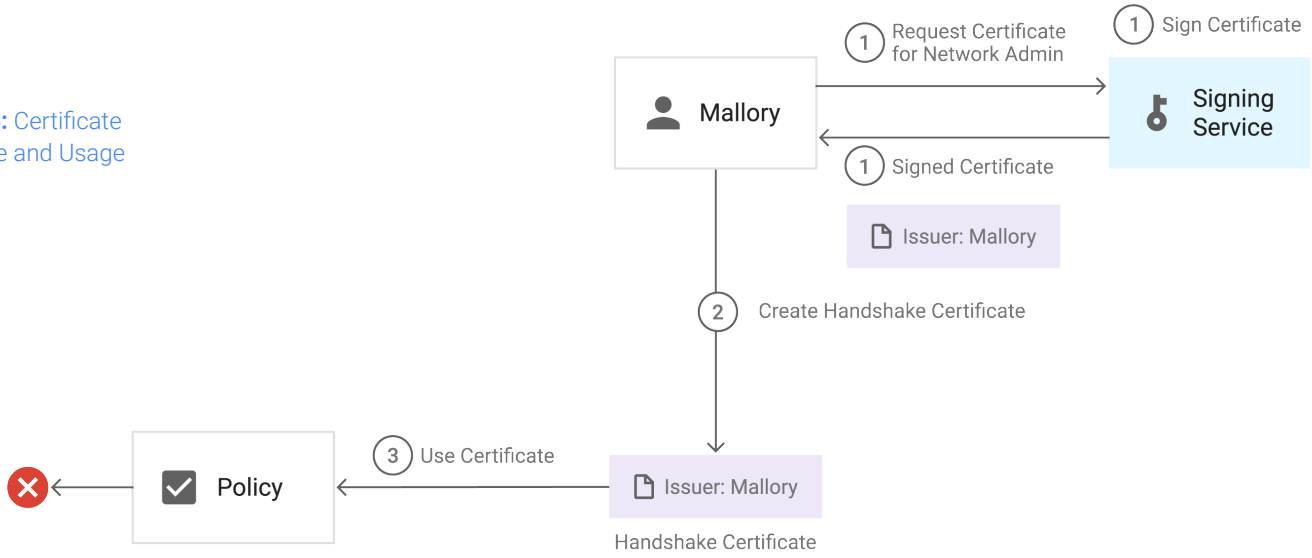


Figure 4: Certificate Issuance and Usage

1. Mallory issues a master certificate for Network Admin identity and gets it signed by the Signing Service. This is similar to the first three steps in Figure 2.
2. Mallory creates and signs a handshake certificate locally for Network Admin, using the master private key associated with the created master certificate.

3. If Mallory tries to impersonate the Network Admin identity by using the created handshake certificate, the ALTS policy enforcer, at the peer that Mallory tries to communicate with, will block the operation.

3.3 Certificate Revocation

At Google, a certificate is invalidated when it expires or it is included in our Certificate Revocation List (CRL). This section describes the design of Google's internal certificate revocation mechanisms, which, at the time of writing this paper, are still undergoing deployment testing.

All certificates issued to human corporate users have a daily expiration timestamp which forces the users to reauthenticate daily. Many of the certificates issued to production machines do not use expiration timestamps. We avoid relying on timestamps to expire production certificates as it can lead to outages caused by clock synchronization issues. Instead, we use the CRL as our source of truth for rotation and incident-response handling of certificates. Figure 5 shows how the CRL operates.

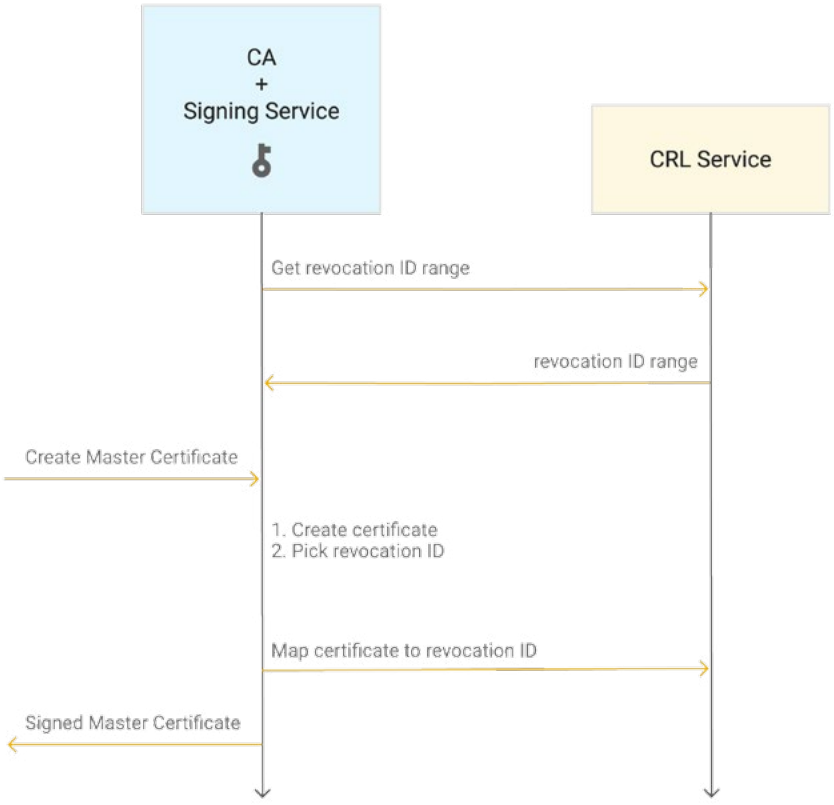


Figure 5: Master Certificate Creation with a Revocation ID

1. When an instance of our CA is initialized[12], it contacts the CRL Service and asks for a revocation ID range. A revocation ID is a 64-bit long ID

[12] In practice, the CA has access to the Signing Service private keys, making the two logical entities as a single physical one.

with two components, an 8-bit certificate category (e.g. human or machine certificate), and a 56-bit certificate identifier. The CRL Service chooses a range of these IDs and returns it to the CA.
2. When the CA receives a request for a master certificate, it creates the certificate and embeds a revocation ID it picks from the range.
3. In parallel, the CA maps the new certificate to the revocation ID and sends this information to the CRL Service.
4. The CA issues the master certificate.

Revocation IDs assigned to handshake certificates depend on how the certificate is used. For example, handshake certificates that are issued to human corporate users inherit the revocation ID of the user's master certificate. For handshake certificates that are issued to Borg workloads, the revocation ID is assigned by the Borgmaster's range of revocation IDs. This ID range is assigned to the Borgmaster by the CRL Service in a process similar to that shown in Figure 5. Whenever a peer is involved in an ALTS handshake, it checks a local copy of the CRL file to ensure that the remote peer certificate has not been revoked.

The CRL Service compiles all revocation IDs into a single file that can be pushed to all Google machines that use ALTS. While the CRL database is several hundred megabytes, the generated CRL file is only a few megabytes due to a variety of compression techniques.

ALTS relies on two protocols: the Handshake protocol (with session resumption) and the Record protocol.

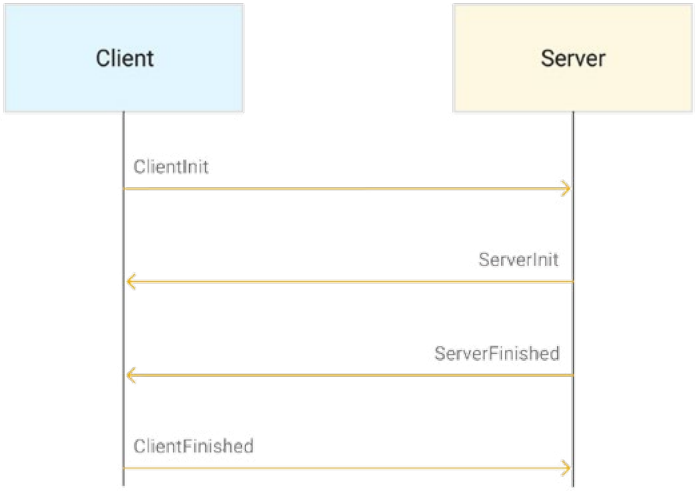
4. ALTS Protocols

ALTS relies on two protocols: the Handshake protocol (with session resumption) and the Record protocol. This section provides a high level overview of each protocol. These overviews should not be interpreted as detailed specifications of the protocols.

4.1 Handshake Protocol

The ALTS handshake protocol is a Diffie-Hellman-based authenticated key exchange protocol that supports both Perfect Forward Secrecy (PFS) and session resumption. The ALTS infrastructure ensures that each client and server have a certificate with their respective identities and an Elliptic Curve Diffie-Hellman (ECDH) key that chains to a trusted Signing Service verification key. In ALTS, PFS is not enabled by default because these static ECDH keys are frequently updated to renew forward secrecy even if PFS is not used on a handshake. During a handshake, the client and server securely negotiate a shared transit encryption key, and the Record protocol the encryption key will be used to protect. For example, the client and server might agree to a 128-bit key that will be used to protect an RPC session using AES-GCM. The handshake

Figure 6: ALTS Handshake Protocol Messages



consists of four serialized Protocol Buffer messages, an overview of which can be seen in Figure 6.

1. The client initiates the handshake by sending a **`ClientInit`** message. This message contains the client's handshake certificate, and a list of the handshake-related ciphers and record protocols the client supports. If the client is attempting to resume a terminated session, it will include a resumption identifier and encrypted server resumption ticket.

2. On receipt of the **`ClientInit`** message, the server verifies the client certificate. If valid, the server chooses a handshake cipher and record protocol from the list provided by the client. The server uses a combination of the information contained in the **`ClientInit`** message and its own local information to compute the DH exchange result. This result is used as an input to Key Derivation Functions[13] along with the transcript of the protocol to generate the following session secrets:
   - A **record protocol** secret key *M* used to encrypt and authenticate payload messages,
   - A **resumption secret** *R* to be used in a resumption ticket in future sessions,
   - An **authenticator secret** *A*.

   The server sends a **`ServerInit`** message containing its certificate, the chosen handshake cipher, record protocol, and an optional encrypted resumption ticket.

3. The server sends a **`ServerFinished`** message containing a handshake authenticator[14]. The value for this authenticator is calculated using a Hash-based Message Authentication Code (HMAC) computed over a pre-defined bit string and the authenticator secret *A*.

4. Once the client receives **`ServerInit`**, it verifies the server certificate, computes the DH exchange result similar to the server, and derives

[13] Specifically, HKDF-Extract and HKDF-Expand as defined in RFC-5869.
[14] ALTS handshaker protocol implementation concatenates ServerInit and ServerFinished messages into a single wire payload.

the same *M*, *R*, and *A* secrets. The client uses the derived *A* to verify the authenticator value in the received **`ServerFinished`** message. At this point in the handshake process, the client can start using *M* to encrypt messages. As the client is now capable of sending encrypted messages, we can say that ALTS has a one RTT handshake protocol.

5. At the end of the handshake, the client sends a **`ClientFinished`** message with a similar authenticator value (see step 3) computed over a different pre-defined bit string. If needed, the client can include an encrypted resumption ticket for future sessions. Once this message is received and verified by the server, the ALTS handshake protocol is concluded and the server can start using *M* to encrypt and authenticate further payload messages.

The Handshake protocol was reviewed by Thai Duong from Google's internal security analysis team and formally verified using the Proverif tool[15] by Bruno Blanchet with the assistance of Martin Abadi.

4.2 Record Protocol

Section 4.1 described how we use the Handshake protocol to negotiate a Record protocol secret. This protocol secret is used to encrypt and authenticate network traffic. The layer of the stack that performs these operations is called the ALTS Record Protocol (ALTSRP).

ALTSRP contains a suite of encryption schemes with varying key sizes and security features. During the handshake, the client sends its list of preferred schemes, sorted by preference. The server chooses the first protocol in the client list that matches the server's local configuration. This method of scheme selection allows both clients and servers to have different encryption preferences and allows us to phase in (or remove) encryption schemes.

4.2.1 Framing

Frames are the smallest data unit in ALTS. Depending on its size, each ALTSRP message can consist of one or more frames. Each frame contains the following fields:

- **Length:** a 32-bit unsigned value indicating the length of the frame, in bytes. This 4-byte length field is not included as part of the total frame length.
- **Type:** a 32-bit value specifying the frame type, e.g., data frame.
- **Payload:** the actual authenticated and optionally encrypted data being sent.

The maximum length of a frame is 1MB plus 4 length bytes. For current RPC protocols, we further limit the frame length as shorter frames require less memory for buffering. Larger frames could also be exploited by a potential

[15] ProVerif: Cryptographic protocol verifier in the formal model.

attacker during a Denial of Service (DoS) attack in an attempt to starve a server. As well as limiting the frame length, we also restrict the number of frames that can be encrypted using the same record protocol secret M. The limit varies depending on the encryption scheme that is used to encrypt and decrypt the frame payload. Once this limit is reached, the connection must be closed.

4.2.2 Payload

In ALTS each frame contains a payload that is integrity protected and optionally encrypted[16]. As of the publication of this paper, ALTS supports the following modes:

- AES-128-GCM, AES-128-VCM: AES-GCM and AES-VCM modes, respectively, with 128-bit keys. These modes protect the confidentiality and integrity of the payload using the GCM, and the VCM schemes[17], respectively.

- AES-128-GMAC, AES-128-VMAC: these modes support integrity-only protection using GMAC and VMAC, respectively, for tag computation. The payload is transferred in plaintext with a cryptographic tag that protects its integrity.

At Google, we use different modes of protection depending on the threat model and performance requirements. If the communicating entities are within the same physical boundary controlled by or on behalf of Google, integrity-only protection is used. These entities can still choose to upgrade to authenticated encryption based on the sensitivity of their data. If the communicating entities are in different physical boundaries controlled by or on behalf of Google, and so the communications pass over the Wide Area Network, we automatically upgrade the security of the connection to authenticated encryption, regardless of the chosen mode. Google applies different protections to data in transit when it is transmitted outside a physical boundary controlled by or on behalf of Google, since the same rigorous security measures cannot be applied.

Each frame is separately integrity protected and optionally encrypted. Both peers maintain both request and response counters, which synchronize during normal operation. If the server receives requests that are out of order, or repeated, cryptographic integrity verification fails, dropping the request. Similarly, the client drops a repeated or mis-ordered response. Furthermore, having both peers maintain the counters (as opposed to including their values in the frame header) saves additional bytes on the wire.

4.3 Session Resumption

ALTS allows its users to resume previous sessions without the need to perform heavy asymmetric cryptographic operations. Session resumption is a feature that is built into the ALTS Handshake protocol.

[16] Payload encryption is negotiated as part of the Record protocol in the handshake.
[17] The 128-bit AES-GCM scheme is based on NIST 800-38D, and AES-VCM is discussed in details in AES-VCM, An AES-GCM Construction Using an Integer-Based Universal Hash Function.

The ALTS handshake allows clients and servers to securely exchange (and cache) resumption tickets which can be used to resume future connections[18]. Each cached resumption ticket is indexed by a Resumption Identifier (ID_R) that is unique to all workloads running with the same identity and in the same datacenter cell. These tickets are encrypted using symmetric keys associated with their corresponding identifiers.

ALTS supports two types of session resumption:

1. **Server side session resumption:** a client creates and encrypts a resumption ticket containing the server identity and the derived resumption secret *R*. The resumption ticket is sent to the server at the end of the handshake, in the **`ClientFinished`** message. In future sessions, the server can choose to resume the session by sending the ticket back to the client in its **`ServerInit`** message. On receipt of the ticket, the client can recover both the resumption secret *R* and the server's identity. The client can use this information to resume the session.

   The ID_R is always associated with a identity and not with specific connections. In ALTS, multiple clients can use the same identity in the same datacenter. This allows clients to resume sessions with servers that they may not have communicated with before, e.g. if a load balancer sends the client to a different server running the same application.

2. **Client side session resumption:** at the end of a handshake the server sends an encrypted resumption ticket to the client in the **`ServerFinished`** message. This ticket includes the resumption secret *R* and the client's identity. The client can use this ticket to resume a connection with any server sharing the same ID_R.

The ALTS handshake allows clients and servers to securely exchange resumption tickets which can be used to resume future connections.

When a session is resumed, the resumption secret *R* is used to derive new session secrets *M'*, *R'* and *A'*. *M'* is used to encrypt and authenticate payload messages, *A'* is used to authenticate **`ServerFinished`** and **`ClientFinished`** messages, and *R'* is encapsulated in a new resumption ticket. Note that the same resumption secret *R* is never used more than once.

5. Tradeoffs

5.1 Key Compromise Impersonation Attacks

By design, the ALTS handshake protocol is susceptible to Key Compromise Impersonation (KCI) attacks. If an adversary compromises the DH private key, or the resumption key, of a workload they can use the key to impersonate other workloads to this workload[19]. This is explicitly in our resumption threat model, as

[18] Session resumption involves lightweight symmetric operations only if ephemeral parameters are not involved.
[19] Key Agreement Protocols and Their Security Analysis.

we want resumption tickets issued by one instance of an identity to be usable by other instances of that identity.

There is a variant of the ALTS handshake protocol that protects against KCI attacks, but it would only be worth using in environments where resumption is not desired.

5.2 Privacy for Handshake Messages

ALTS is not designed to disguise which internal identities are communicating, so it does not encrypt any handshake messages to hide the identities of the peers.

5.3 Perfect Forward Secrecy

Perfect Forward Secrecy (PFS) is supported, but not enabled by default, in ALTS. We instead use frequent certificate rotation to establish forward secrecy for most applications. With TLS 1.2 (and its prior versions), session resumption is not protected with PFS. When PFS is enabled with ALTS, PFS is also enabled for resumed sessions.

5.4 Zero-Roundtrip Resumption

TLS 1.3 provides session resumption that requires zero roundtrips (0-RTT), however this has weaker security properties[20]. We decided not to include a 0-RTT option in ALTS because RPC connections at Google are generally long-lived. Consequently, reducing the channel setup latency was not a good tradeoff for the additional complexity and/or reduced security that 0-RTT handshakes require.

6. Further References

For information on how Google encrypts data in transit, see our Encryption in Transit in Google Cloud whitepaper.

For an overview of how security is designed into Google's technical infrastructure, see our Google Infrastructure Security Design overview.

[20] Replay Attacks on Zero Round-Trip Time: The Case of the TLS 1.3 Handshake Candidates.