Vidal, I., et al. SCoT: A secure content-oriented transport, in: *Journal of Network and Computer Applications, Vol. 105, March 2018, pp. 63-78*

DOI: https://doi.org/10.1016/j.jnca.2018.01.001

SCoT: A Secure Content-oriented Transport

Ivan Vidal[1,], Jaime Garcia-Reinoso[1], Ignacio Soto[1], Francisco Valera[1], Diego Lopez[1]

[a]*Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid, Avda. Universidad, 30, 28911 Leganés (Madrid), Spain*
[b]*Telefónica I+D, C/ Zurbarán 12, 28010 Madrid, Spain*

Abstract

The evolution of the Internet has resulted in the deployment of new application-level solutions to enhance the scalability and efficiency of content dissemination (e.g., content delivery networks and peer-to-peer systems). However, despite of this improvement on performance, the utilization of this type of solutions introduces new security concerns, as a content provider must necessarily delegate the role of distributing the content to third parties, and current security solutions, such as TLS and IPsec, do not allow authenticating the original content provider or the content itself in these scenarios. In this paper, we present SCoT, a transport-layer protocol that allows a content provider to bind protection to content, enabling content authentication at receivers regardless of any third party infrastructures that have been used to disseminate the content. Content authentication procedures are executed transparently to end-user applications. We implemented a fully operational prototype of the protocol in Java, including an API to support the development of SCoT applications. We utilized it to configure an experimentation scenario that served to validate a theoretical analysis of the SCoT throughput and to illustrate the performance that can be achieved in a practical deployment. The paper concludes describing a set of use cases of the protocol.

Keywords: Transport security, content authentication, content distribution

*Corresponding author

1. Introduction

In recent years, the massive use of the Internet, with its large number of geographically distributed users, has resulted in the proliferation of new application-level mechanisms to support the scalable and efficient access to content. Well known successful examples of these mechanisms are Content Delivery Networks (CDN), peer-to-peer (P2P) systems and mirror sites. However, being the communication model in the Internet based on host-to-host communications, the utilization of this type of solutions inevitably entails the dissociation of the role of delivering the content from the publisher of that content, as contents will be distributed from server equipment that does not belong to the provider (e.g., a CDN server). This creates a challenge for the authentication models currently used in the Internet to guarantee the provenance of the content, as these allow authenticating the server participating in a communication, typically using TLS, but not the provider of the content or the content itself, which would be most desirable in case of networked applications that delegate the role of content delivery. A workaround to deal with this issue is sharing authentication credentials, such as the private key of the content provider, with a third party (in particular, with the server delivering the content). However, this is highly undesirable from the point of view of security, as it breaks the fundamental principle in authentication that the possession of a private key, which enables the computation of digital signatures, is only limited to the key owner. Some other solutions are being proposed, which are reviewed in Section **??**.

In this paper, we present SCoT, a transport layer protocol to protect the distribution of content in the Internet, enabling the authentication of a received content independently of any intermediate equipment that have been used to distribute it. The design of the mechanisms for content protection in SCoT has been inspired on the concept of content-based security (**?**), where security is bound to the content being transmitted as opposed to network security solutions of common use in the Internet (e.g., TLS or IPsec), which protect the distribution of information by securing the end-to-end communications used to deliver the content.

When a content provider uses the SCoT protocol to deliver a solicited information (e.g., a file, a video stream, etc.) to an interested consumer, the SCoT layer at the provider equipment generates a sequence of cryptographic information elements, known as SCoT credentials. These credentials are streamed towards the consumer, as they are generated, along with the

content itself. At the consumer side, credentials are used by the SCoT layer to authenticate the information as it is received (receiving all the information is not required to authenticate it). The generation of SCoT credentials and content authentication procedures are executed transparently to communicating applications, both at the sender and the receiver sides. Additionally, receivers can securely re-distribute a given content, by instructing their SCoT layer to re-use the credentials generated by the provider.

Being a transport-layer solution, SCoT can work with other Internet protocols and services operating at different layers of the TCP/IP protocol stack. On the one hand, these include the diverse protocols that are available at the application-layer. As an example, a server could use the HTTP protocol to distribute content to interested receivers, and utilize SCoT at the transport-layer to authenticate the distribution of this content. On the other hand, SCoT can work in cooperation with other transport and network-layer security-related solutions, such TLS and IPsec, to address the authentication challenges related with content dissemination in a variety of relevant use cases. Examples of these use cases, where the utilization of SCoT may be of particular interest, include the distribution of content through CDNs, installation of software components and upgrades, the communication of information under the new IoT paradigm, and the secure dissemination of data in the novel and proliferating softwarization environments offered by cloud and network function virtualization infrastructures.

We have developed a fully operational prototype implementation of SCoT. In the paper, we use this prototype to obtain experimental results showing the performance of the protocol, which may be in general comparable to that of TCP, although with the limitations imposed by the execution of the cryptographic operations that are needed to support content authentication procedures (e.g., creating and verifying SCoT credentials).

The rest of the paper is organized as follows. Section **??** reviews related work. Section **??** describes the basic operation of SCoT and Section **??** provides a detailed description of the different procedures in the protocol. Section **??** analyzes the performance of the protocol, both analytically and experimentally with the prototype implementation. We describe a set of relevant use cases of SCoT in Section **??**. Finally, Section **??** summarizes the main conclusions and future directions of our work.

2. Current solutions for content protection

Security is not built in the core of the Internet, and several extensions have been proposed to protect content. In this sense, there are different alternatives to provide end-to-end security like the IPsec set of specifications, which provide authentication and encryption services at the IP layer (**?**), or the recent TCPcrypt (**?**) specifically focused on the TCP layer and implemented as additional TCP options for both encryption and authentication. But it is probably the Transport Layer Security (TLS) protocol (**?**) the one that has been most widely adopted in the Internet. Born as the Secure Sockets Layer (**?**), it was closely related in its origins to the World Wide Web in order to be able to deliver secure Web content. Now TLS is an application agnostic protocol that aims at establishing a secure communication channel (providing confidentiality, integrity and authentication) to exchange data. TLS is capable of authenticating the server that is providing the corresponding content, and optionally the client, and afterwards encrypting all the data that both entities may transmit by performing a secret key exchange that will be used by the confidentiality service.

However, considering the evolution that content distribution schemes have taken nowadays, far from the centralized WWW model, nodes may end up authenticating the hosts that serve the content, but not the content providers that own the content or the content itself.

Some solutions have appeared in order to address scenarios where content authentication is required, but content is provided by an intermediate entity, e.g., through a secondary server or a relay. SCD (Secure Content Delegation) (**?**) for instance, combined with blind caches (**?**), presents a proposal to be able to serve content using HTTP from secondary servers (caches) but with the added value of integrity and confidentiality provisioning and source authentication based on TLS. SCD basically allows direct TLS connections to the main server, but the response redirects the client to the secondary server where the real content can be found. The content provider can balance the amount of contents that are directly served or are delegated to the caches (blind caches in case the content is encrypted). The content provider also includes hashes of the different contents in its answers so that later on they can be matched with the content downloaded from the caches, and integrity can be guaranteed. However, blind caches have been shown to present privacy issues and, in any case, SCD requires having the main server available to retrieve the content, as well as the reception of the whole content to verify

its integrity and authentication.

There are other specific proposals for CDNs like the Custom Certificate and Shared Certificate mechanisms. In the Custom Certificate mechanism, content providers share their private key with CDN providers (which is undesirable from the point of view of security); and in the Shared Certificate approach, CDN provider's Certificate Authority issues an X.509 certificate to the CDN provider, including in the Subject Alternative Name (SAN) the content provider domain name. Authors in (**?**) describe the main drawbacks of both Customer and Shared Certificates mechanisms. The authors also present a solution based on DNS resolution but, as all solutions based on TLS, it tackles the problem of privacy and data integrity between two end points, but not the authentication of the content itself.

There are other alternatives to support content authentication in CDNs. In (**?**), the authors modify the typical TLS key exchange mechanism performed between the client and the secondary server, allowing the secondary server to be identified using the content provider certificate. The client will obviously use this certificate to protect the communication with the secondary server but this one has no private key to continue the exchange. The secondary server will instead forward the message to the content provider that will return the proper information so as to complete the handshake. A recent proposal along this idea is the Delegated Credentials for TLS (**?**), where a TLS server operator can issue its own credentials within the scope of a certificate issued by an external CA. This allows the server to terminate TLS connections on behalf of the certificate owner.

LURK (Limited Use of Remote Keys) is another recent initiative fostered within the IETF, addressing authentication issues of current content distribution models in the Internet. In the context of LURK, several approaches are being explored. One approach to prevent abuse of delegated credentials, a potential problem that exists in other solutions such as Custom Certificate, is to control their validity, e.g., by providing limited life credentials. Other interesting approaches have been proposed in LURK like the Session Key Interface for TLS (**?**), or the PFS-preserving protocol (**?**) where the role of the TLS server is split into the Edge Server (that in fact terminates the TLS tunnel) and the Key Server (or Key Owner), being this second one the responsible for all private key operations as requested by the Edge Server. In (**?**), the author follows a similar approach introducing LURK notation like the LURK server for the key holder, LURK client for the edge server ("client" of the LURK service) and LURK Administrator (new entity that

controls the relation between the client and the server).

Other research initiatives address the fundamental challenges of content dissemination in the Internet using more disruptive approaches. Information Centric Networking (ICN) (**?**) is a salient initiative in this category. As opposed to the host-centric approach used in existing packet networks, ICN redefines the Internet architecture to operate in terms of named data, supporting the efficient and secure access to content with the extensive use of in-network caching and innovative security concepts. In this respect, rather than addressing content protection by establishing secure point-to-point communications between publishers and consumers, as in the current Internet, ICN uses a different approach where protection is inherently bound to content. As an example, CCN (**?**) (a prominent solution under the umbrella of ICN) authenticates the association between every piece of content and its corresponding name using digital signatures, which are transported along with the content through the network. This concept, denominated as content-based security in CCN, is fundamental to our solution, as we will see in the following section.

Figure **??** relates the different proposals, including SCoT, and summarizes their main characteristics:

- Content authentication capabilities. The ability of each solution to provide content authentication. This is the main objective of SCoT, but some protocols can only achieve this by key or identity delegation, or are not providing content authentication at all.

- Caching of authenticated content. This column defines if authenticated content can be served from intermediate entities independently of the availability of the content provider server.

- Independence from applications. This information specifies if the solution can be used by different applications, i.e., if it is application agnostic. This is only possible for solutions located below the application layer.

- Functions needed in different equipment. This column indicates in which type of equipment (routers and/or endpoints) each solution needs specific functionality.

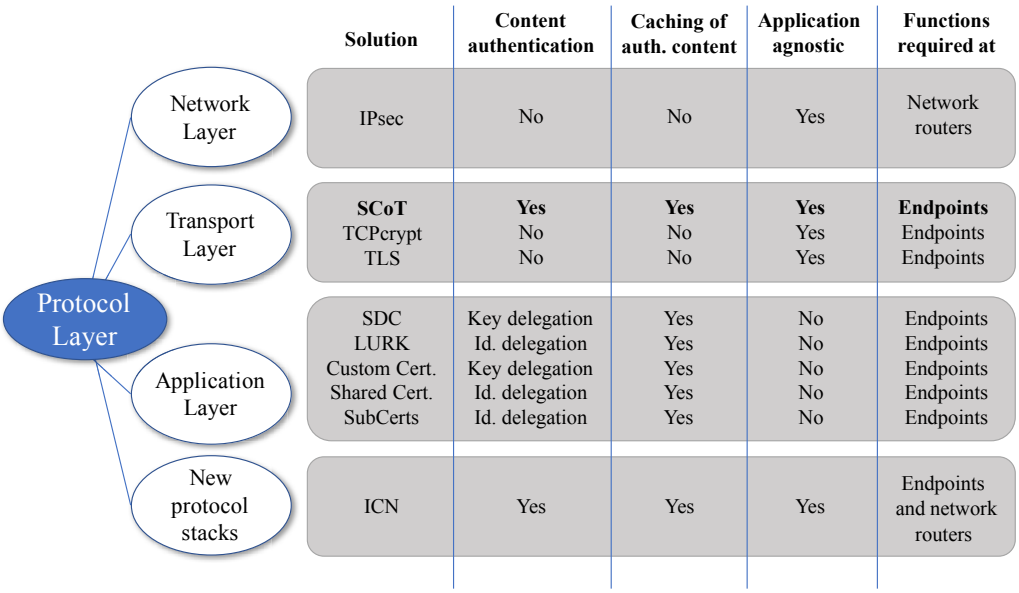

Figure 1: Summary of current solutions and SCoT

3. Operation of the SCoT protocol

This section describes the basic operation of SCoT. For clarity, SCoT procedures are presented in the context of an illustrative example, considering an entity that uses the proposed solution to retrieve a specific content (i.e., a file) from a server owned by the content provider. This entity then relays the content to an interested consumer. The distribution of content to the relay and to the consumer is protected using SCoT, being the content authenticated as it is received at both receiver endpoints.

In our solution, content is named by its corresponding provider, and it is divided into blocks as it is provided by the sender application. The association between each block of data and the content name is authenticated using digital signatures, which are generated by the SCoT layer of the provider equipment transparently to the application. The SCoT protocol distributes the data blocks along with the information that is needed to authenticate them. Authentication is carried out by the SCoT layer as the blocks of data are received, transparently to the end-user applications, which can determine the validity of the received content according to the level of trust established on the provider that has generated the signatures.

3.1. Establishment of a SCoT connection

Figure **??**.a shows the exchange of protocol data units (PDUs) that are required to deliver the requested file from a server application, running at the provider equipment, to a client application that is executed at the content relay. SCoT is a connection-oriented protocol, hence the delivery of the file requires the previous establishment of a connection between the server and the client applications. Similarly to TCP, the SCoT protocol supports the bidirectional transmission of a stream of data in both directions of a SCoT connection.

As a first step in our example, the client application triggers the SCoT layer to establish a connection with the server application, indicating the domain name of the provider equipment (or its corresponding IP address) and the SCoT transport port where the server is listening to incoming connections (SCoT ports can be allocated independently of other transport protocol ports, such as TCP or UDP). The SCoT layer completes the connection setup with a two-way handshake, based on the exchange of a *CONNECT* PDU sent from the client side (step 1 in Fig. **??**.a), and a *CONNECTED* PDU, which is used by the server side to confirm the flawless establishment of the connection (step 2). Our protocol operates over TCP, which guarantees the reliable ordered exchange of SCoT PDUs between the communicating endpoints. The procedures related with the management of SCoT connections are detailed in Section **??**.

3.2. Activation of security contexts and content delivery

Once the connection has been established, content can be exchanged between the client and the server application. SCoT operates at the transport layer, hence it can be used to distribute information from any application-level protocol. In our example, the server application is an HTTP server capable of distributing a number of files, which are stored at the provider equipment, using the HTTP protocol. To request the delivery of one of these files, the client application issues an HTTP GET request, specifying the URL of the file, and uses the SCoT connection to send the request to the server. In the content-oriented model followed by our solution, the HTTP request is considered to be a piece of content that must be delivered from the relay to the HTTP server. Taking this into account, before transmitting the request, the client application informs the SCoT layer to activate a security context for this HTTP content. Security contexts represent a fundamental

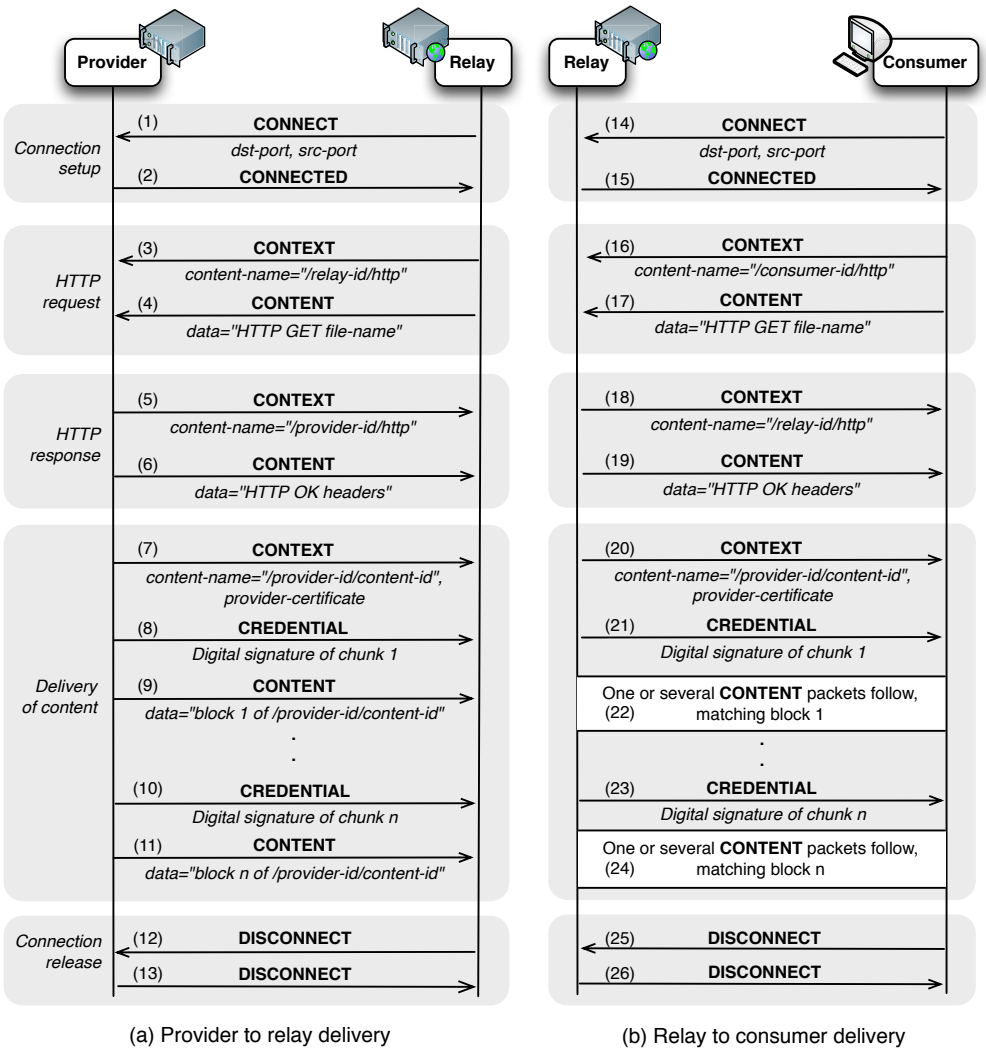

Figure 2: Exchange of SCoT PDUs

concept in SCoT. A security context includes the status information that is needed to unambiguously identify the content that is to be delivered, as well as to execute the mechanisms that guarantee its appropriate authentication. Detailed information on security contexts is provided in Sect. **??**.

We want to observe that, in the considered example, files are the fundamental information items that the content provider desires to protect, hence

their authentication is mandatory at receiver equipment (e.g., at the relay). Conversely, the authentication of the HTTP messages, which are issued to request and encapsulate those files, is not really needed to guarantee the successful operation of the file distribution service. For this reason, the security context activated by the client application will only include the specification of the name that will be assigned to HTTP content (*'/relay-id/http'* in Fig. **??**.a), with no additional information that enables authentication procedures. The activation of the security context at the client side is signaled to the SCoT layer of the server side by means of the *CONTEXT* PDU shown in step 3 of Fig. **??**.a, causing the activation of the context at the provider equipment. The client application can now send the HTTP request as a stream of bytes over the SCoT connection. The SCoT layer encapsulates and sends the received stream as a *CONTENT* PDU (step 4), and delivers the stream as it is received to the server application. According to the security context associated with HTTP content, the stream of bytes is sent unprotected through the network. The security context may be re-utilized to send subsequent HTTP messages, from the client to the server application, as long as the SCoT connection remains open. For this purpose, information about active security contexts is maintained at the SCoT layer of the client and server endpoints during the lifetime of the connection, and *CONTENT* PDUs transport a content identifier, i.e., a fixed size identifier that is derived from the content name determined by the security context, which allows associating the PDU with its corresponding security context.

Upon receiving the HTTP request, the server application sends the requested file to the relay encapsulated in an HTTP OK response. To do this, the server application first activates a security context for HTTP content. Analogously to the HTTP request, as the protection of the HTTP headers is not needed, the security context will only contain an application-specific name assigned to HTTP content (*'/provider-id/http'* in step 5 of Fig. **??**.a). The server application can now start the transmission of the HTTP response as a stream of bytes over the SCoT connection, which is delivered to the client application being transmitted as *CONTENT* PDUs by the SCoT transport layer.

Prior to the transmission of the requested file in the body of the HTTP response (i.e., after sending the headers of the HTTP response in step 6 of Fig. **??**.a), the server application activates a second security context. This context will include the provider-specific content name assigned to the file, along with a certificate of the content provider, containing the public key of

the provider that the server application wishes to use for content authentication purposes. After the successful activation of the context at both ends of the SCoT connection (step 7 in Fig. **??**.a), the server application starts the transmission of the file as a stream of data. According to a set of configuration parameters specified by the security context, the SCoT layer of the provider will fragment the stream of data into blocks. Each block will be used to generate a SCoT credential for that block, including an identifier of the block of data, a fixed-length representation of the block and a digital signature, which authenticates the association between the block of data and its corresponding identifier. Credentials are generated transparently to the server application, computing the digital signatures with the private key corresponding to the certificate within the security context (detailed information about data delivery and credentials is provided in Sect. **??**). After being generated, each credential is appended to the security context at the provider equipment, being then sent as a *CREDENTIAL* PDU to the other end of the SCoT connection (steps 8 and 10 in Fig. **??**.a).

The SCoT layer of the relay verifies the validity of the credentials as they are received, being this process transparent to the client application. A successful signature verification means a positive declaration from the owner of the certificate that the association between a block of data and its corresponding identifier, as they are received in a credential, is valid. Upon flawless verification, the credential is appended to the active security context for subsequent utilization. After receiving a *CONTENT* PDU in steps (9) and (11) in Fig. **??**.a, the SCoT layer of the relay verifies if the block of data enclosed in the PDU corresponds to the block of data validated by the credential. If so, the block is considered as valid, according to the credentials provided by the provider. The client application can always retrieve a reference to an active security context during an ongoing SCoT connection, enabling it to access the provider-specific name of the received content as well as the identity of the provider that has generated the credentials (this information is included in the certificate of the security context). With this information, the application can determine the validity of the received content, according to the level of trust established on the provider that has generated the credentials. A failing credential verification, or a block of data that cannot be matched with a valid credential, is notified to the client application, which is in charge of taking any appropriate actions to handle the error condition (typically terminating the SCoT connection). The interface offered by the SCoT layer to end-user applications is described in Sect. **??**.

3.3. Termination of a SCoT connection

After completing the file transfer, the client application at the relay uses the reference to the security context to store a copy of this context at the application layer (e.g., as a local file in a hard drive). As we will see shortly, this is a fundamental step to guarantee the subsequent secure dissemination of the received content from the relay to any interested consumers.

Finally, the client application at the relay triggers the SCoT layer to terminate the SCoT connection, causing the transmission of a *DISCONNECT* PDU towards the provider equipment (step 12 in Fig. **??**.a). Analogously to TCP, the process to terminate a SCoT connection is performed separately by the server and the client applications. Accordingly, when the SCoT layer of the relay receives a *DISCONNECT* PDU from the provider (step 13), it releases the resources utilized at the transport layer (particularly, the information about any security contexts activated during the connection) and the SCoT connection is closed.

3.4. Relay of content

Continuing with our example of operation, let us assume that an interested consumer now requests the delivery of the file stored in the content relay. Figure **??**.b shows the exchange of SCoT PDUs that are required to transfer the file from a server application, running at the relay equipment, to a client application that is executed at the consumer equipment. As it is observed in the figure, the procedure starts analogously to the previous case, involving a connection setup between the client and the server application, the establishment of security contexts for HTTP content and the exchange of an HTTP request, indicating the URL of the solicited file, and the corresponding HTTP response encapsulating the file (steps 14 to 19 in Fig. **??**.b). Upon verifying the availability of a security context for the content that is to be delivered (a copy of this security context has been stored in a local file), the relay application retrieves the context from the file and triggers the SCoT layer to activate it. Consequently, as shown in step (20) of Fig. **??**.b, a *CONTEXT* PDU is transmitted to the other end of the SCoT connection, including the name of the content that is to be transferred along with the certificate that will be used for content authentication procedures (i.e., the certificate of the content provider). With this information, the security context is also activated by the SCoT layer of the consumer equipment. The server application at the relay can now send the requested file as a stream

of bytes over the SCoT connection, being transmitted by the SCoT layer as *CONTENT* PDUs (steps 22 and 24 in Fig. **??**.b).

Credentials stored in the security context, which were originally generated by the SCoT layer of the provider equipment, are sent as *CREDENTIAL* PDUs (steps 21 and 23) as they are needed for content authentication purposes at the consumer equipment (details are provided in Sect. **??**). This enables to deliver credentials to the receiver endpoint making a conservative use of bandwidth resources. Content authentication proceeds as in the previous case, where the file was transferred from the provider to the relay equipment, and the consumer application can determine the validity of the received content based on the level of trust established on the provider that has generated the credentials. After the successful file transfer, the consumer application requests the termination of the SCoT connection (steps 25 and 26).

4. Detailed SCoT procedures

This section complements the information on the operation of SCoT, as provided in the previous section, with detailed information on data structures and on the main procedures that comprise the definition of the protocol.

4.1. The SCoT-sockets API

The Application Programming Interface between the SCoT layer and the applications is called SCoT-sockets. The SCoT-sockets interface, similarly to TCP sockets, allows establishing a connection, sending and receiving the data corresponding to a specific content, and closing a connection. SCoT-sockets also provide access to functionality not present in TCP, such as setting a security context and content authentication. Table **??** details the complete set of operations made available by the SCoT-socket API to end user applications.

4.2. SCoT PDU types and format

The format of a SCoT Packet Data Unit (PDU) is shown in Fig. **??**. The header has a fixed length of 16 bytes. The two initial fields in the header are the source and destination ports (16 bits each), which identify the endpoints exchanging the PDU. The next field is the Type of PDU (8 bits), which indicates which PDU is being exchanged. The types of PDUs defined until now in SCoT are shown in Table **??**. Then there is a reserved field for future extensions of the protocol. The last field of the header is the Length

Operation	Description
Activate security context	Activates a security context at the SCoT layer of a sender application, signals the activation to the other end of the SCoT connection.
Retrieve security context	Obtains a reference to the active security context. A receiver application can use this reference to access the fields of the context that are relevant for content authentication purposes, such as the name of the content that is being received and the certificate of the content provider (if available).
Read bytes	Reads data from the stream of bytes corresponding to the content specified by the security context. The operation does only retrieve authenticated data to the calling application. Notifies any error condition to the receiver application (e.g., a credential or a content block failing the authentication process).
Read immediately	Reads any available data from the stream of bytes corresponding to the content specified by the security context. The operation may return not-yet-authenticated data to the calling application. Notifies any error condition to the receiver application.
Write bytes	Transmits a number of bytes, corresponding to the content determined by the security context, as a stream of data to the other end of the SCoT connection.
Transmit immediately	Solicits the immediate transmission of any data waiting to be processed in the send buffer of the SCoT layer of the sender application.
Establish connection	Establishes a SCoT connection.
Terminate connection	Terminates a established SCoT connection.

Table 1: Operations available to applications in the SCoT-sockets API

SCoT PDU	Use
CONNECT	Connection establishment (see section **??**)
CONNECTED	Connection establishment (see section **??**)
CONTEXT	Reports the activation of a security context to the peer of the SCoT communication (see section **??**)
CONTENT	Includes a chunk of application data (see section **??** and section **??**)
CREDENTIAL	Includes a digital signature authenticating a chunk of data (see section **??** and section **??**)
DISCONNECT	Connection release (see section **??**)

Table 2: SCoT PDUs

field (64 bits) that is the length in bytes of the SCoT PDU (including the header). It is needed because SCoT must be able to identify the boundaries of the exchanged PDUs (a service that TCP does not provide). Note that the length of SCoT PDUs can be very large, as TCP will segment them if needed to adapt to potential restrictions due to the MTU.

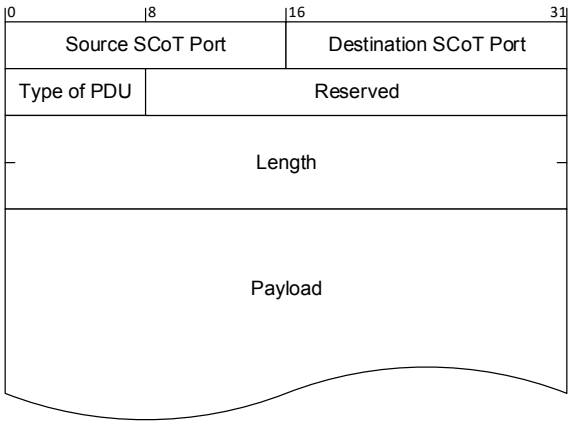

Figure 3: SCoT PDU format

4.3. Connection management

SCoT is a connection oriented transport protocol. Therefore, applications that want to communicate using SCoT must first open a SCoT connection

and, in the same way, they need to close the connection after using it. In this section, we are going to describe the management of connections in SCoT.

In an open SCoT connection, the end-point SCoT-sockets are identified by the source and destination IP addresses, and the source and destination SCoT ports (similarly to TCP). As mentioned before, SCoT ports are independent of ports in other protocols and allow the identification of the SCoT-sockets involved in a SCoT communication. The SCoT PDUs include the source and destination SCoT ports of the respective connection (see Fig. **??**), so the SCoT layer in the reception side can use this information to associate the payload of the PDU to the right SCoT connection.

To provide a complete description of connection management in SCoT, we must also look at how SCoT layers in different end systems exchange SCoT PDUs. SCoT operates over TCP and requires a well-known or assigned TCP port. A SCoT layer has a permanent TCP socket waiting for connection requests in this well-known TCP port. Any SCoT layer, to reach a remote SCoT layer, uses this port as destination TCP port.

The states of a SCoT connection, in a client and in a server, are shown in Fig. **??**. A server application creates a SCoT-socket to wait for incoming connection requests. This SCoT-socket will be associated to a local SCoT port. While the SCoT-socket is waiting for incoming connection requests, it is only identified by the local SCoT port and we say that the SCoT connection is in "Listen" state (it is similar to the "Listen" state in TCP).

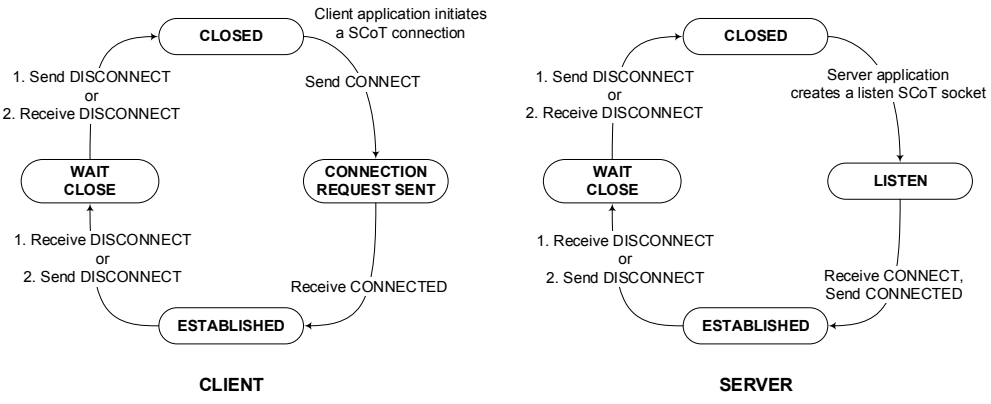

Figure 4: Connection states in SCoT

A client application, to communicate with some remote server using

SCoT, creates a SCoT-socket indicating the destination IP address and destination SCoT-port. The source SCoT-port can be any one, as the server will learn it, to be able to reply, from the request. The SCoT layer in the client machine will create a TCP socket to communicate with the SCoT layer in the server machine. Once there is a TCP connection between the SCoT layers in the client and server machines, the SCoT layer in the client sends a CONNECT PDU to the server. Note that, as we are using TCP, the sending of SCoT PDUs is reliable and ordered, so no special measures to deal with losses or out-of-order data are needed. In the server machine, the SCoT layer identifies the destination SCoT port in the CONNECT PDU. If there is no application waiting for connections in that port, a DISCONNECT PDU is sent to the source. If a server application is waiting for connections in that port, a new SCoT-socket is created for the connection and the connection moves to "Established" state in the server. The SCoT layer also sends a CONNECTED PDU to the client. When the CONNECTED PDU is received in the SCoT layer in the client, the respective connection moves from "Listen" state to "Established" state, and the client application can start using the socket to send and receive authenticated data.

Regarding the closing of connections, client and server applications close their side of a SCoT connection independently. Closing the connection in one side means that that side has finished sending data, but it still can receive data (similar to TCP). The SCoT layer sends a DISCONNECT PDU to inform the other side of the situation. When a SCoT layer has sent and received the DISCONNECT PDU for a connection, the SCoT connection is closed and the respective resources can be released.

4.4. Security contexts

In SCoT, the protection to be applied to a given piece of content is defined by a security context, a concept that is fundamental to our protocol, as security associations and sessions to IPsec and TLS, respectively. A security context is formally defined as the status data that are required at the SCoT layer to: (1) identify the content that is to be transmitted; and (2) to support appropriate authentication procedures for that content both at sender and receiver endpoints. A sender application must always activate a security context at the SCoT layer of the sender endpoint before transmitting content towards a receiver. This is notified to the SCoT layer of the receiver application, enabling the activation of the security context at both sides of the SCoT connection and guaranteeing a coherent configuration of

the authentication procedures. This way, a security context allows protecting a unidirectional stream of data delivered from the sender to the receiver application (a receiver application will need to activate appropriate security contexts to protect content transmitted in the reverse direction). Several security contexts may be activated during the lifetime of a SCoT connection, to support the delivery of multiple content objects even belonging to different content providers.

In our solution, similarly to CCN, content is named by its corresponding provider, who may use any desired namespaces and conventions to support content-to-name allocations. As an example, a content provider *'provider.com'* could allocate the name *'/provider.com/videos/title0514/H264-2500k'* to a video content from its catalogue, 'title0514', compressed with the H.264 video codec to a bit rate of 2.5 Mb/s. These content names are nevertheless opaque to the SCoT protocol, being their structure and semantics only meaningful in the context of provider and consumer applications.

The format of a security context is described in Fig. **??**. It includes the name assigned to the content to be protected and may include a certificate of the content provider, which serves to identify the pair of public and private keys that will be used to generate and verify the digital signatures that support content authentication. The provider certificate is an optional field, and may be absent in case that content authentication is not required by a sender application. The security context also contains a set of additional variables that enable to parametrize the operation of the protocol (providing the flexibility to accommodate diverse application requirements), as well as keep track of the dynamic status values required by transmission and reception processes. Finally, a security context may include a number of credentials, which are generated and appended to a context as content is delivered to interested consumers.

4.5. Delivery of content

As previously commented, SCoT is a connection-oriented protocol that provides a reliable end-to-end communication service. After the establishment of a SCoT connection between two communicating applications, each of them can use the SCoT-sockets API to independently activate a security context and request the transmission of a byte stream, i.e., the content identified by the context, to the other end of the SCoT connection.

As data is sent by an application through a SCoT-socket, the SCoT layer places the data into a send buffer. The protocol breaks data in the send

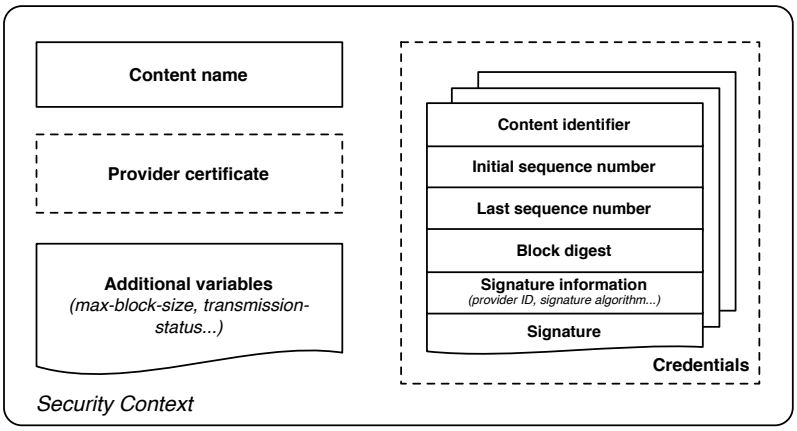

Figure 5: Structure of a security context and a SCoT credential

buffer into content blocks, which are sequentially processed, being the size of each block determined by a transport-level Maximum Block Size (MBS). Figure **??** represents the procedures taking place at the SCoT layer of a sender application to process a content block. As it can be observed in the figure, each content block is used to generate a credential, which is delivered to TCP for transmission as a CREDENTIAL PDU (step 1 in Fig. **??**) The credential will serve to protect the delivery of the content block, which is transmitted as a CONTENT PDU using TCP (step 2 in Fig. **??**). In SCoT, a credential is always sent prior to its corresponding block of data. This provides a higher degree of protection, as it guarantees the receiver that the information needed for subsequent data verification is always available.

The precise structure of a SCoT credential is detailed in Fig. **??**. A credential identifies the content block that is to be protected, using a fixed-length representation of the name corresponding to the content being transmitted (this name is determined by the security context), along with an identification of the first and the last byte of the content block within the byte stream transmitted by application (for this purpose, SCoT maintains a sequence number that increases by one for each byte received through the SCoT-socket). The fixed-length representation of the content name (referred to as content identifier in the figure) is made with a hash function, aiming at limiting the size of a credential and hence its corresponding transmission overhead. A credential contains a hash value of the content block (i.e., the block digest in the figure) and additional supporting information, including

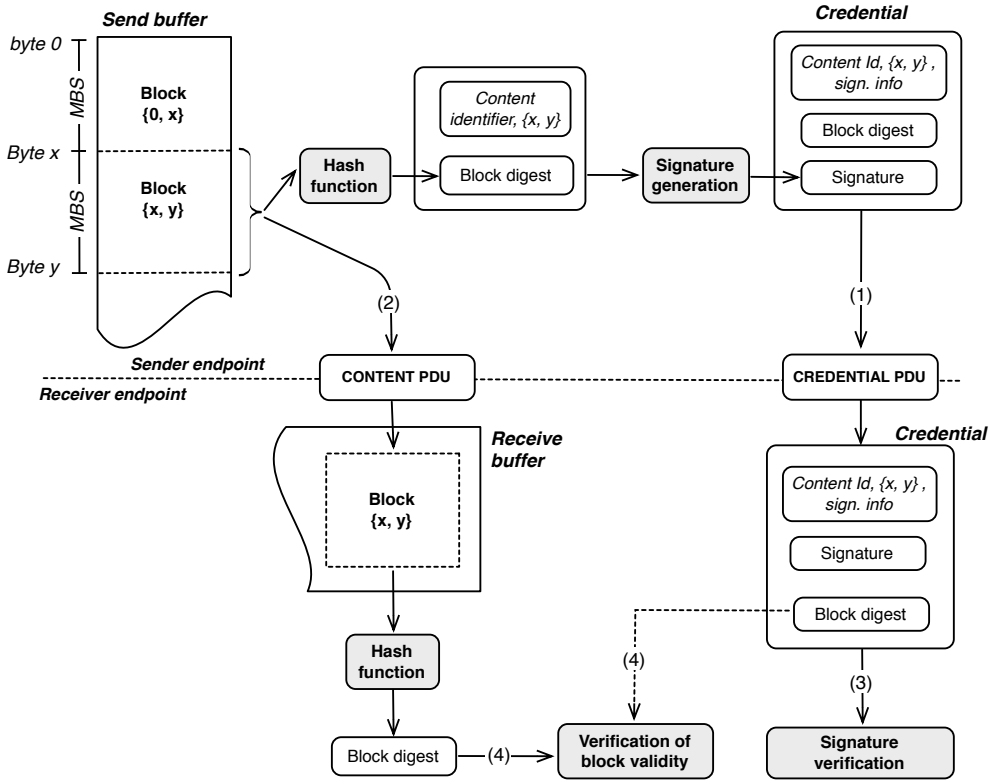

Figure 6: Authentication procedures at sender and receiver endpoints

an identification of the content provider and other security-related data that will be used for authentication purposes. Finally, the credential contains a digital signature of the previous fields, computed with the private key of the content provider[1] (i.e., the private key corresponding to the certificate indicated in the security context). This way, a credential provides a signed declaration from a content provider that a byte range of a given content, as determined by the content identifier and the initial/last sequence numbers provided in the credential, correspond to the content block identified by the

[1]An alternative would be to negotiate a session key and use symmetric cryptography to authenticate the exchange. This approach would have advantages from the point of view of computational resources, but it is not valid for SCoT, because a relay with a symmetric key would be able to use it to generate or modify the content and the respective credentials, without the approval of the content provider.

block digest. The digital signature allows authenticating the credential itself, avoiding potential attacks consisting of injecting bogus credentials in a SCoT communication.

As the size of the last content block may be lower than the MBS, SCoT does not process it immediately. Instead, it keeps it in the buffer until new data is provided by the application through the SCoT-socket and additional MBS-byte blocks can be created and processed. To satisfy any application-specific requirements, the SCoT-sockets API also allows an application to solicit the immediate transmission of the data stored in the send buffer, which may result in the transmission of content blocks smaller than MBS bytes (see Sect. **??**).

We want to highlight that the value of the MBS may be dependent on the application scenario. As an example, a file transfer application may use relatively large values of the MBS to keep the bandwidth overhead low (due to credentials and PDU headers), while avoiding an excessive consumption of network resources due to the complete download of a bogus file. On the contrary, a video streaming application may require lower values of the MBS, to play authenticated blocks of video without incurring in excessive application-layer playout delays. For this reason, our solution allows applications to specify an appropriate value for the MBS as a parameter of the security context that is to be activated. Consequently, the MBS used by SCoT is not related to the maximum segment size (MSS) of TCP nor the IP MTU, i.e., a content block in SCoT can be spread into several TCP segments, in case that MBS is larger than the MSS, or a TCP segment may include data from several content blocks, as in the case of a MBS lower that the MSS.

Figure **??** also shows the processing of CREDENTIAL and CONTENT PDUs at the receiver endpoint. Upon receiving a credential, the SCoT layer of the receiver application authenticates it, verifying its corresponding digital signature. The verification of the signature is done using the public key of the content provider, which is contained in the provider certificate stored in the security context that has been activated prior to data delivery. A successful signature verification means that the credential has been generated by the content provider, so the credential is stored in the active security context. The content block encapsulated in the CONTENT PDU coming right after a credential is placed, as it is received, into a receive buffer (the content PDU is received as a byte stream according to the transmission service of TCP). When the reception of the PDU is completed, the SCoT layer validates the

content block in the receive buffer, matching a hash value of the block against the block digest contained in the credential. If both hash values match, the authentication of the content block is successful and it can be consumed by the receiver application.

As indicated in Sect. **??**, the receiver application can use the SCoT-sockets API to read a stream of authenticated data from the receive buffer. Alternatively, the application can read data from the receive buffer that has not yet been authenticated, to satisfy any existing application-level requirements. In this case, the receiver application can check the information available in the security context to verify which received data have been authenticated at the SCoT layer and can therefore be consumed at the application level. In any case, the reception of a credential or a content block that fails the SCoT authentication procedures causes the notification of this error situation to the receiver application, which can then terminate the SCoT connection and execute any application-layer procedures that might be appropriate (e.g. notify the end user about the error condition).

4.6. Re-use of existing credentials

In our solution, the SCoT layer of the receiver application verifies the credentials provided by the sender, appending them to the active security context in case of a successful validation. This process is executed transparently to the receiver application, which can retrieve a reference to the active security context at any point in time during the lifetime of the SCoT connection. This reference can then be used to store a copy of the security context at the application layer, either in application-specific memory space or in persistent storage (e.g., as a local file in a hard drive). This will enable to securely relay the received content to other interested consumers.

When an application receives a request over a SCoT connection to transmit a specific content, which has previously been received by this or another application running at the same equipment, the relay application retrieves a copy of the security context from the local storage[2] and activates the context at the SCoT layer, using the SCoT-sockets API. The relay application can then send the solicited content as a stream of bytes through the SCoT-socket. The data stream is delivered by the SCoT layer to the other end of

[2]Note that to support the relay of content received by another application at the same host, the security context must necessarily be kept in a shared storage

the connection encapsulated in CONTENT PDUs. It is important to highlight that the relay application may use any transmission pattern to deliver the requested content to the SCoT layer. Hence CONTENT PDUs do not necessarily match the PDUs that were originally transmitted to deliver the content to the relay equipment. Credentials stored in the security context, which were originally generated by the SCoT layer of the provider equipment, are sent as CREDENTIAL PDUs. Each credential is sent prior to the first byte of the data block that was used to issue the credential. This way, credentials are always available at the receiver for appropriate content authentication, despite receiving relayed content from an intermediate equipment not belonging to the content provider.

The proposed approach avoids implementing a local storage of security contexts at the transport layer, as well as complex replacement strategies that guarantee the availability of a security context to retransmit a specific content provided by a relay application. Instead, a relay application willing to securely deliver a given content simply needs to maintain a copy of the security context that contains the transport-layer information required by SCoT to support its secure distribution (e.g., content identification and credentials). Analogously, the original sender application, causing the generation of the credentials, may also store a copy of the security context, which may be later retrieved and activated to support the secure delivery of the same content, avoiding the re-generation of credentials and amortizing computing costs for subsequent requests of the same file.

4.7. Security properties of the protocol

As already discussed, SCoT has been designed to address the authentication challenges inherent to the application-level mechanisms commonly used in the Internet to support the efficient and scalable distribution of content, which typically dissociate of the role of delivering the content from the publisher of that content (i.e., the content provider). Concretely, SCoT offers a transport-layer solution that enables the authentication of a received content independently of any intermediate entities that have been used to relay that content.

With this purpose, a sender application using SCoT allocates provider-specific names to content, and the SCoT layer of the sender provides the cryptographic mechanisms to bind these names to their corresponding content, relying on SCoT credentials that include a digital signature made with

the private key of the content provider, which is never exposed nor delegated to third parties. The SCoT layer of the receiver authenticates the credentials and the corresponding content as they are received. The receiver application can then examine the provider-specific name of a received content, identify it as the expected information, and determine its validity according to the level of trust established on the entity that has generated the credentials. As an example, an application receiving a content named as */provider.com/videos/title0514/H264-2500k*, could identify this content as valid as long as it satisfies SCoT cryptographic verification processes and the credentials have been generated by *'provider.com'*.

Applications using SCoT may be diverse, including not only those that require the dissemination of static information (e.g., web pages or prerecorded video), but also applications that distribute dynamically generated content (e.g., live video). For this reason, SCoT only implements functionalities that are expected to be common to all the applications of the protocol. As an example, the design of the protocol intentionally does not support the verification of the content as a whole (e.g. the verification of an entire file in a file download application), as this feature is not required in the case of applications that generate and distribute dynamic information nor in those where content has to be consumed while it is still being received (e.g. applications that stream stored video).

Content is exchanged as a byte stream between sender and receiver applications, being this stream divided by the SCoT layer into blocks of data that are processed separately. A SCoT credential is always sent in advance to its corresponding content, and includes an identification of the first and the last byte of the block of data within the byte stream. This way, the protocol can detect anomalous situations, such as the reception of content blocks with incorrect size (e.g., truncated or extended blocks) or out-of-sequence blocks (i.e., starting with an incorrect first byte identifier). The protocol can also detect the insertion of outdated versions of a content block within the byte stream, providing that communicating applications use specific conventions to distinguish different versions of the same content (e.g., maintaining a version identifier in content names). In either case, anomalous or error conditions are notified to the receiver application, which can take any appropriate actions to handle the reported issue (e.g., discarding erroneous blocks of content and continuing with the reception of subsequent pieces of the same content, or terminating the SCoT connection).

Finally, we want to highlight that SCoT does not authenticate entities

that disseminate content (e.g., content providers or intermediate entities acting as content relays). This function is already covered by existing security protocols and solutions, such as TLS or IPsec, and SCoT has not been designed as a replacement for these solutions. On the contrary, it can work in coordination with them. As an example, if a given content is requested by an interested consumer using the HTTP protocol, as in the example considered in Section **??**, and the authentication of the source of the HTTP messages is required by the receiver application, a feasible alternative to protect the dissemination of the content could be using SCoT over TLS, this way supporting both the authentication of the received content (provided by SCoT) and the authentication of the entity that sends the content encapsulated as HTTP messages (provided by TLS). Additionally, the utilization of TLS in this example would allow encrypting the transmitted content, guaranteeing the confidentiality of the content exchange between sender and receiver applications (an alternative option could be to implement the encryption at the application layer).

4.8. Considerations about a standalone SCoT transport

In this paper we have considered a SCoT protocol that exchanges PDUs using the communication service provided by TCP. This simplifies many of the SCoT procedures, as we can relay on the service provided by TCP. However, a standalone implementation of SCoT, working directly over IP, is also possible. A standalone implementation would have potential advantages in terms of efficiency and flexibility. For example, in a standalone implementation we could extend SCoT to offer two different services, one reliable (TCP-like) and one best effort (UDP-like). A specification of a standalone SCoT protocol is out of the scope of this paper but the aspects that would need to be addressed are:

1. Reliability: an error control mechanism to detect and retransmit lost PDUs.
2. Segmentation: Without affecting the concept of MBS (the length of the block of data that is authenticated), SCoT must avoid, segmenting PDUs at the SCoT layer if needed, the fragmentation by IP of IP datagrams that transport SCoT PDUs.
3. Flow control: a mechanism to protect resources in the SCoT receiving entity.

Term	Description
S_{pdu}^{Cr}	Average size of a CREDENTIAL PDU
S_{header}^{Co}	Average size of the header of a CONTENT PDU
t_c	Average time required to compute a SCoT credential
t_v	Average time required to authenticate a SCoT credential
t_b	Average time required to authenticate a block of received data using its corresponding credential
MBS	Maximum block size of SCoT
Th_{TCP}^{cong}	Maximum average throughput that can be achieved by the SCoT layer over a TCP connection, as determined by the available network bandwidth

Table 3: Summary of terminology

4. Congestion control: a mechanisms to avoid creating congestion in the network. In this case, it is important to keep compatibility with the TCP congestion control mechanism.

5. Validation of the proposal

5.1. Description of SCoT throughput

This section presents a macroscopic description of the throughput that can be achieved by the SCoT protocol, with the objective of identifying the parameters that are of particular relevance in the performance provided by the protocol to SCoT applications. In our analysis, we assume a baseline scenario where a sender application transmits an arbitrarily large file to a receiver application using the SCoT protocol. This file is delivered as a stream of bytes using the underlying TCP connection that is established between the SCoT layer of the sender and the receiver endpoints. As a reference, table **??** describes the terminology utilized in this section.

As it was commented in Sect. **??**, the SCoT layer of the sender application places the incoming data (the byte stream corresponding to the file) into a send buffer. The protocol divides these data into blocks of MBS bytes (i.e., the maximum block size of SCoT), and processes the content blocks sequentially. Each of these blocks is used to generate a credential, using a digital

signature algorithm. The credential and the content block are transmitted as SCoT PDUs using the underlying TCP connection. More concretely, if we denote t_c as the average time that is needed to compute a credential, then the SCoT layer of the sender can deliver a CREDENTIAL and a CONTENT PDU to TCP for transmission with an average period of t_c seconds.

Let S_{pdu}^{Cr} be the size of a CREDENTIAL PDU in bytes. We denote S_{header}^{Co} as the size in bytes of the header of a CONTENT PDU. Let Th_{TCP}^{cong} be the maximum average throughput that can be achieved by the steady-state TCP connection, as determined by TCP congestion control mechanisms according to the network bandwidth that is available between the equipment hosting the sender and receiver applications. With this, the transmission rate offered to TCP by the SCoT layer of the sender can approximately be expressed as:

$$Th_{TCP}^{tx} = \min\{\frac{S_{pdu}^{Cr} + S_{header}^{Co} + MBS}{t_c}, Th_{TCP}^{cong}\} \quad (1)$$

That is, the transmission rate is given by the value of the MBS and the time required to process it to generate its corresponding SCoT credential. In case that this rate exceeds the maximum throughput that can be made available by the underlying TCP connection, the TCP send buffer gets full and the SCoT layer necessarily decreases the sending rate to Th_{TCP}^{cong}.

In our analysis, we assume that the receiver application continuously reads data as they are available from the receive buffer of the SCoT connection. In turn, the SCoT layer of the receiver places data in that buffer as they are made available by TCP, except in those time periods where it executes the authentication procedures of received credentials and content blocks. In case that these authentication procedures do not allow to remove data from the TCP receive buffer at the rate given by equation **??**, then the TCP receive buffer will eventually become full. Consequently, the transmission rate of the sender will be limited by the rate at which the SCoT layer of the receiver retrieves data from TCP, according to TCP flow control mechanisms. This rate can be calculated under the observation that, as the TCP receive buffer is full, a CREDENTIAL PDU and its corresponding CONTENT PDU are always available for processing by the SCoT layer of the receiver. Therefore, SCoT will continuously be reading PDUs from TCP and executing the corresponding authentication procedures.

We denote t_v as the average time that is needed to authenticate a SCoT credential, and t_b as the average time required to authenticate its corresponding block of data (i.e., the time delay to compute a hash value of the received

block and match this value against the block digest contained in its corresponding credential). The maximum average rate at which SCoT can retrieve data from TCP can then be expressed as:

$$Th_{TCP}^{rx} = \frac{S_{pdu}^{Cr} + S_{header}^{Co} + MBS}{t_v + t_b} \tag{2}$$

Considering equations **??** and **??**, the maximum average TCP throughput that can be achieved by the SCoT layer of the sender can be expressed as:

$$Th_{TCP} = \min\{Th_{TCP}^{tx}, Th_{TCP}^{rx}\} \tag{3}$$

From equation **??**, and considering that the receiver application can only retrieve a content block of MBS bytes for each CONTENT PDU received at the SCoT layer, the average throughput that can be achieved by the SCoT connection, when credentials have to be generated, can roughly be expressed as:

$$\begin{aligned} Th_{SCoT} &= \frac{MBS}{S_{pdu}^{Cr} + S_{header}^{Co} + MBS} \cdot Th_{TCP} \\ &= \min\{\frac{MBS}{t_c}, \frac{MBS}{S_{pdu}^{Cr} + S_{header}^{Co} + MBS} \cdot Th_{TCP}^{cong}, \frac{MBS}{t_v + t_b}\} \end{aligned} \tag{4}$$

Let us now consider a scenario where an intermediate entity, which has received the file using the SCoT protocol, relays it to an interested consumer. For simplicity, we assume that the sender application at the relay provides the stream of bytes corresponding to the file to the SCoT layer, using the same value of MBS that was configured by the original sender application. With these considerations, the SCoT layer at the relay will break the incoming byte stream into blocks of MBS bytes. For each of these blocks, it will use the underlying TCP connection to transmit a CREDENTIAL PDU, encapsulating the credential that was originally generated by the sender application in the provider equipment, along with a CONTENT PDU, containing the actual content block validated by the credential.

As the generation of credentials is not needed in this case, and under the assumption that the received application retrieves data from the SCoT layer as they are available, the average TCP throughput that can be achieved by SCoT is given by the minimum value between Th_{TCP}^{cong} and Th_{TCP}^{rx}. Analogously to the previous case, given that the SCoT layer places a content

block of MBS bytes in the receive buffer of the SCoT connection for each received CONTENT PDU, the average throughput that can be achieved by the receiver application, when credentials are re-used, can be approximated as:

$$\begin{aligned} Th'_{SCoT} &= \frac{MBS}{S^{Cr}_{pdu} + S^{Co}_{header} + MBS} \cdot \min\{Th^{cong}_{TCP}, Th^{rx}_{TCP}\} \\ &= \min\{\frac{MBS}{S^{Cr}_{pdu} + S^{Co}_{header} + MBS} \cdot Th^{cong}_{TCP}, \frac{MBS}{t_b + t_v}\} \end{aligned} \tag{5}$$

5.2. Prototype implementation

To validate the proposed solution, we developed a software prototype of the SCoT protocol. This prototype was implemented in Java 1.8 and includes two main components, the SCoT-sockets API and the SCoT daemon. The design of the SCoT prototype is schematized in Fig. **??**.

The SCoT-sockets API has been developed as a Java package, and provides the implementation of the different operations described in Table **??**. The API allows programming end-user Java applications that support the protected distribution of information through the SCoT protocol, using socket abstractions similar to those used in TCP. Additionally, it supports the operations related with the creation and authentication of SCoT credentials. Digital signatures are generated and verified using the RSA algorithm (cryptographic operations are supported in our prototype with the utilization of the Java Cryptography Architecture[3]). Every instance of the SCoT sockets API communicates with the local SCoT daemon using an internal TCP connection, for the purposes of initiating, accepting and terminating SCoT connections, as well as exchanging CONTEXT, CREDENTIAL and CONTENT PDUs. The *internal dispatcher* is the component of the SCoT daemon in charge of (1) handling the establishment and termination of internal TCP connections between different instances of the SCoT-sockets API and the SCoT daemon; and (2) maintaining the external TCP connections that are needed to support the exchange of SCoT PDUs with remote endpoints. The *external server* is the component of the SCoT daemon that handles incoming

[3]Java Cryptography Architecture (JCA) Reference Guide: https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html

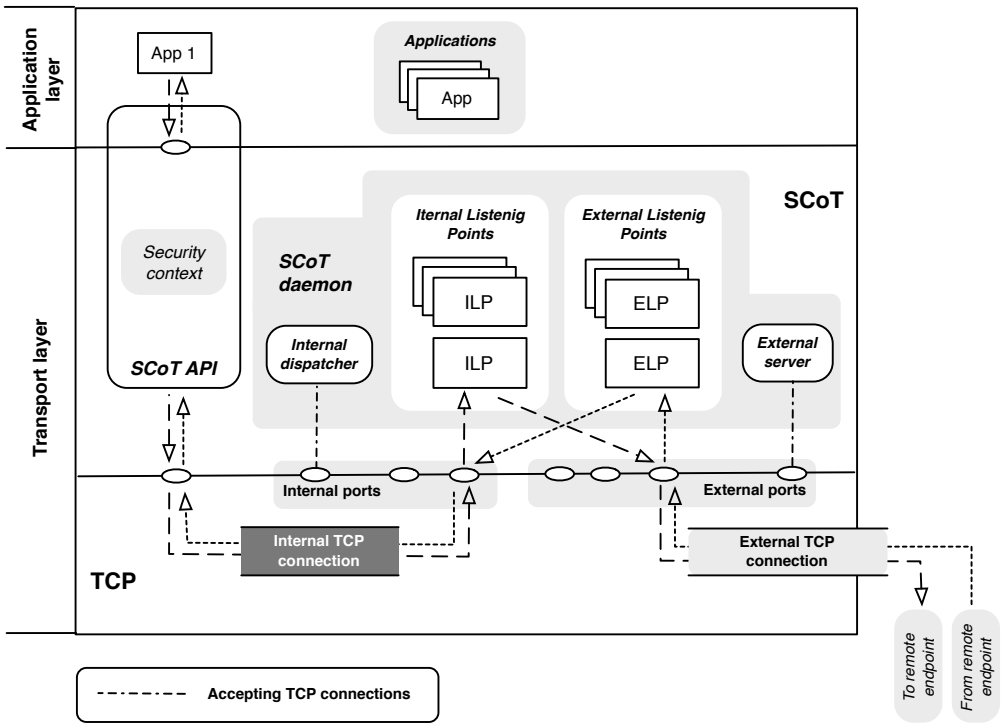

Figure 7: Overview of the prototype SCoT implementation

TCP connections from remote peers. Both the *internal dispatcher* and the *external server* are executed by the SCoT daemon as Java Threads.

The establishment of a SCoT connection results in the instantiation of a pair of *internal listening point* and *external listening point* at each peer of the connection. The *internal listening point* handles the reception of SCoT PDUs and other relevant information from the SCoT sockets API involved in the connection, and uses the external TCP connection to forward protocol PDUs towards the communication peer. The *external listening point* is in charge of receiving SCoT PDUs from the remote peer of the SCoT connection, and forwarding them to the appropriate SCoT-sockets API as specified by the destination ScoT port of each PDU header.

5.3. Practical evaluation

To validate out macroscopic analysis of the SCoT throughput, as well as to illustrate the performance that can be achieved by the protocol in a

practical deployment, we have carried out a set of experiments using our prototype implementation.

We implemented a file sharing service based on the HTTP protocol. The service includes a client application, which can request the delivery of a file stored in a remote equipment, issuing an HTTP GET request that specifies the URL of the file. A simple HTTP server at the remote equipment processes the HTTP request and encapsulates the requested file in an HTTP response that is sent back to the soliciting client. The client application and the HTTP server have been implemented using the SCoT-sockets API, which allows the distribution of the HTTP content and the requested file using the SCoT protocol, according to the diagram of Fig. **??**. Both software components were deployed in two off-the-shelf computers (CPU Intel Core2 Quad Processor 2.66 GHz and 4 GB memory) interconnected by a 100 Mb/s Ethernet switch.

In our experiments, we used the client application to retrieve a file of 300 MB from the HTTP server, using the following values for the SCoT Maximum Block Size (MBS): 500 B, 1 KB, 5 KB, 10 KB, 50 KB, 100 KB, 500 KB and 1 MB. In a first stage, the HTTP server was configured to activate a new security context for every solicited file, causing the generation and the transmission of SCoT credentials to the client application for each file downloaded. For each value of MBS, we carried out 30 executions of the client application to download the file from the HTTP server. With these executions, we measured the following variables for each value of the MBS: (1) the average throughput provided by the SCoT protocol to the client application; and (2) the average time required to compute a SCoT credential, i.e., t_c; (3) the average delay to authenticate a SCoT credential, t_v; and (4) the average delay to authenticate a block of received data using its corresponding credential, t_b.

In a second stage, we configured the HTTP server to store a copy of every new security context as a local file in the hard drive. This way, after processing the first request for a given file, the generated security context can be retrieved and activated to serve subsequent requests of the same file, hence avoiding the cost corresponding to the re-generation of SCoT credentials, as these are already included in the stored security context. With this configuration, for each considered value of the MBS, we carried out 30 executions of the client application to download the 300 MB file from the HTTP server. Each set of 30 executions was preceded by an initial request to download the file, causing the creation and storage of the security context

Variable	Value
S_{pdu}^{Cr}	221 B
S_{header}^{Co}	13 B
Th_{TCP}^{cong}	93.7 Mb/s

Table 4: Values for the theoretical calculation of the SCoT throughput

to be later re-utilized in the following executions with the same MBS. With the 30 executions of the client application, we measured, for each value of the MBS, the average throughput provided by SCoT to the client application, along with the values of t_v and t_b.

Figure **??** represents the average throughput provided by SCoT to the client application for each value of MBS, considering the case where the SCoT layer at the server is required to generate new SCoT credentials for each file request (first stage of the experiments), as well as the case where security contexts are stored and credentials are re-used for successive downloads of the same file (second stage of the experiments). In order to validate the macroscopic analysis of the SCoT performance, the graph also illustrates the maximum average throughput that can be achieved with the utilization of our protocol, given by equations **??** and **??**. For the theoretical calculation of the values given by these equations, we used the average values of t_c, t_v and t_b measured in our experiments for each value of MBS. The values of the other variables needed for the calculation, which are independent of the MBS, are presented in Table **??**[4].

Focusing on the case where SCoT credentials are generated for each new file download request arriving at the HTTP server (first stage of the experiments), the figure shows that for values of MBS up to 10 KB, the SCoT throughput is mainly limited by the time required to compute a SCoT credential, i.e., t_c, increasing with the value of MBS. For values of MBS greater or equal to 50 KB, the transmission rate offered by the HTTP server to the SCoT layer increases to the point where the SCoT throughput gets limited

[4]The values of S_{pdu}^{Cr} and S_{header}^{Co} were obtained from our prototype implementation. The value of Th_{TCP}^{cong} was estimated running the *iPerf* tool (https://iperf.fr) between the server and the client endpoints.

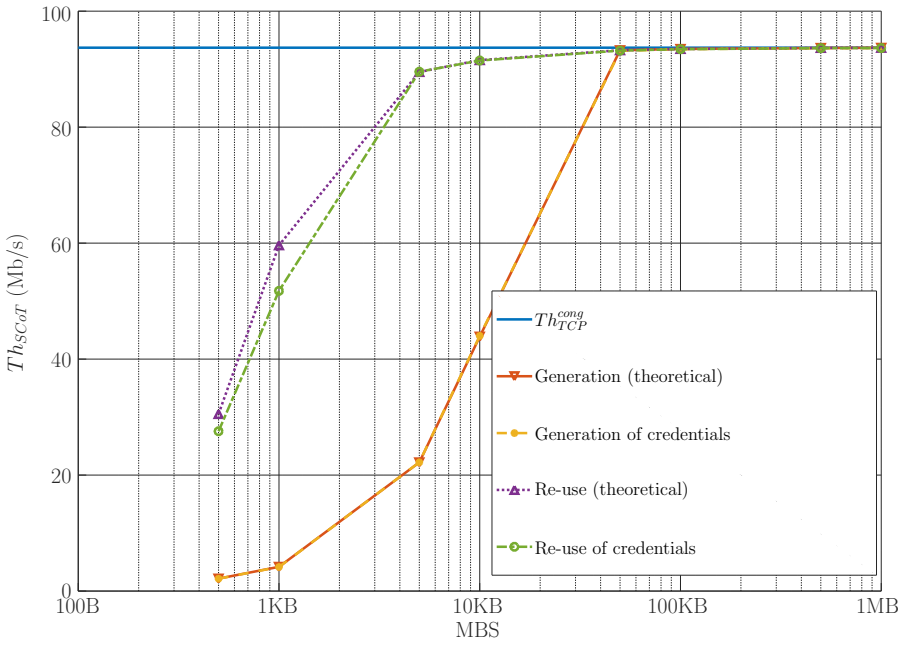

Figure 8: Experimental and theoretical SCoT throughput

by the maximum average TCP throughput given by Th_{TCP}^{cong}. The resulting SCoT throughput is a proportion of the TCP throughput, due to the overhead of the SCoT CONTENT headers and the CREDENTIAL PDUs (as given by the second argument of the *min* function of equation **??**.)

In case that SCoT credentials are re-used (second stage of the experiments), we can observe in the figure that, for values of MBS up to 1 KB, the SCoT throughput is limited by the maximum average rate at which the SCoT layer of the client application can retrieve data from TCP (this rate given by equation **??**). Higher values of the MBS allow increasing the SCoT throughput to the point of being limited by a proportion of the maximum average TCP throughput given by Th_{TCP}^{cong}, as determined by the first argument of the *min* function of equation **??**.

While decreasing the value of MBS augments the granularity of content authentication and enables the prompt detection of bogus content, our experiments indicate that this comes at a cost of increased overhead, as it increases the number of CREDENTIAL PDUs and CONTENT headers that need to

be transmitted. Moreover, for low values of MBS, the SCoT throughput may be limited by the time delays needed to execute cryptographic operations related to the generation and verification of SCoT credentials. However, we want to emphasize that, even using moderately low values of MBS, SCoT may provide appropriate performance metrics for the transmission and relay of content. As an example, considering our experiments with an MBS value of 50 KB, the protocol operates with a very limited overhead (the relation between MBS and $S_{pdu}^{Cr} + S_{header}^{Co} + MBS$ is approximately 99.5%), achieving an average throughput comparable to TCP.

For both stages of experimentation, practical results corroborate the theoretical values of throughput that can be achieved by SCoT, validating the correctness of the macroscopic analysis previously presented. With this observation, we used the results of our analysis to gain a better understanding on the performance of a steady-state SCoT connection under different values of available network bandwidth. For this purpose, we considered the following values of Th_{TCP}^{cong} (i.e., the average throughput that can be achieved by TCP between the sender and the receiver endpoints): 1 Mb/s, 10 Mb/s, 100 Mb/s and 1 Gb/s. For each of these values, we calculated the average throughput that can be provided by SCoT to a receiver application as the value of the MBS increases[5]. Figure **??** represents the obtained results, considering the case where SCoT generates a SCoT credential for each data block of MBS bytes. On the other hand, Fig. **??** covers the case where the sender re-uses existing credentials to protect the transmission of each content block. For the sake of clarity, the graphs shown in both figures are normalized, representing the relation between the throughput that can be provided by SCoT and the average TCP throughput (Th_{TCP}^{cong}).

Focusing on the case where credentials are generated (Figure **??**), for moderately low values of the average TCP throughput (1 Mb/s), the SCoT throughput is a proportion of Th_{TCP}^{cong}, as given by the second term of equation **??**. The reason for this is that the generation of SCoT credentials allows providing a transmission rate to TCP higher than the average throughput that can be made available by the underlying TCP connection. Therefore, the transmission rate is limited to Th_{TCP}^{cong} at the TCP layer, and the SCoT

[5]For each considered value of the MBS, the values of t_c, t_v and t_b needed for the calculations were obtained from our previous experimental results (the values of these variables only depend on the MBS and on the performance characteristics of the end-user hardware platforms).

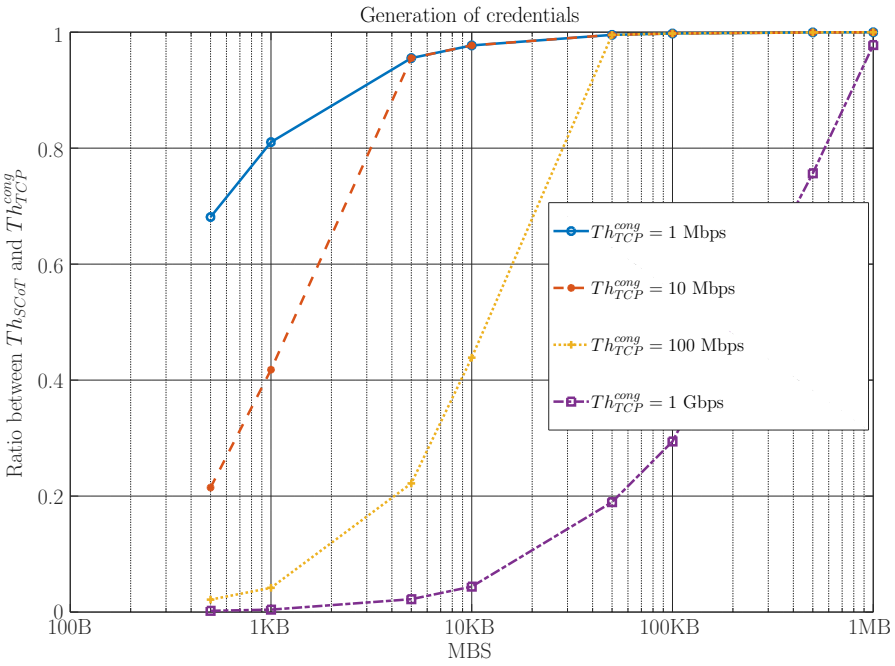

Figure 9: Effect of the available bandwidth on the SCoT throughput (credentials are generated)

throughput is given by the proportion of application-layer data that is transported over the TCP connection (e.g., approximately 68% of Th_{TCP}^{cong}, in case that $Th_{TCP}^{cong} = 1$ Mb/s and MBS = 500B). When the average TCP throughput increases to 10 Mb/s, the SCoT throughput for low values of MBS (500B and 1000B) is limited by the time delay required to compute a SCoT credential (t_c), being determined by the first term of equation **??**; for higher values of MBS (above 5 KB), the transmission rate allowed by the generation of credentials exceeds the maximum TCP throughput, hence the SCoT throughput is given by the corresponding proportion of Th_{TCP}^{cong}. As the maximum average TCP throughput increases, the value of MBS that allows maximizing the SCoT throughput, and obtaining similar performance to TCP, also increases. However, we want to highlight that even in case of high available bandwidth, the MBS that allows maximizing the SCoT throughput typically remains below reasonable limits (e.g., 1 MB for $Th_{TCP}^{cong} = 1$ Gb/s). This allows providing similar performance to TCP with a reasonable consumption

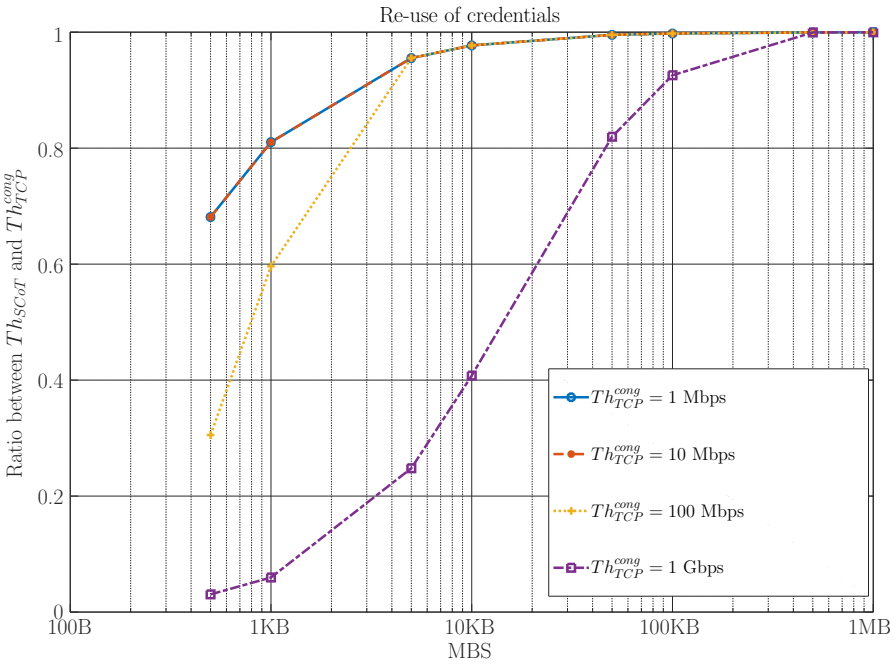

Figure 10: Effect of the available bandwidth on the SCoT throughput (credentials are re-used)

of memory resources and without adding significant processing delay both at the sender and the receiver endpoints.

In case where credentials are re-used, CREDENTIAL and CONTENT PDUs are sent to the receiver endpoint at the maximum throughput that can be made available by the TCP layer. Therefore, the SCoT throughput will be in principle given by a proportion of the average TCP throughput, as determined by the first term of equation **??** (this is observed for values of Th_{TCP}^{cong} of 1 Mb/s and 10 M/s). However, as the maximum TCP throughput increases with the available bandwidth, the frequency of cryptographic operations, needed to verify SCoT credentials and content blocks, also increases. This may limit the rate at which the SCoT layer reads data from the TCP receive buffer, consequently reducing the transmission rate of TCP and the throughput offered by SCoT to the receiver application, which will then be given by the second term of equation **??** (this can be seen for values of Th_{TCP}^{cong} of 100 Mb/s and 1 Gb/s). Analogously to the case where SCoT

credentials are generated, we want to emphasize that even for very large values of Th_{TCP}^{cong}, the MBS that allows maximizing the SCoT throughput can be utilized with an affordable cost in terms of memory resources (e.g., 1 MB for $Th_{TCP}^{cong} = 1$ Gb/s) and processing delay.

With respect to the size of a security context, which is a parameter of interest in our solution (contexts need to be stored at the application layer to support the re-use of SCoT credentials), its value is determined by the number of SCoT credentials stored in the context. This number can be calculated as the ratio between the size of the downloaded file and the value of the MBS (a SCoT credential is generated for every MBS bytes block of the file). As a credential presents a fixed size (221 B) in our prototype implementation, the size of a security context practically decreases inversely proportional to the MBS, being almost negligible with respect to the file size for values of MBS above 50 KB. This is shown in Fig. **??**.

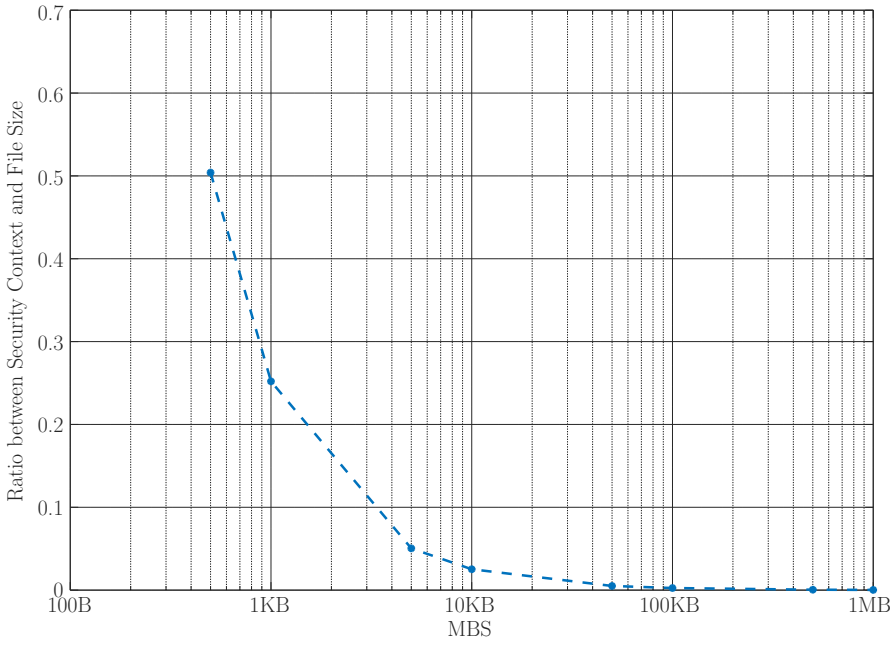

Figure 11: Size of a security context with respect to the size of the transferred file

6. Use case scenarios

This section presents several use case scenarios where SCoT can be applied, showing the advantages of using our proposal compared with the software tools used nowadays in such services.

6.1. Content Delivery Networks

Figure **??** shows the general architecture of a Content Delivery Network (CDN). Content providers, who own the content to be distributed (e.g. live or prerecorded video, multimedia content linked in social networks, etc.), select a CDN provider to distribute their content to end-users (clients). The main objective of content providers using a CDN is to increase the quality of experience perceived by their clients, pushing their content to replica servers near those end-users. CDNs are also useful to handle attacks, like denial of service (DoS).

Content providers move their content from their origin servers to the CDN Replica servers (backend communication). The CDN may distribute the delegated content to other replica servers, which are located near the clients, following a push approach. It is also possible to use a pull approach, where the content is transferred from origin to replica servers as a consequence of a client request. The proper behavior depends on the algorithms used by the Content Manager to distribute content among replica servers.

When a client requests a particular content delegated by the content provider to a CDN provider (frontend communication), the Redirector of the CDN replies back to the client with information about the proper replica server storing the solicited content. The Redirector selects the best replica server based on content availability, replica servers load, proximity between replica servers and clients, etc.

In a traditional web service not using CDN providers, clients directly access origin servers. The transmission of the content only involves these two entities, so it is feasible to use protocols like the Hypertext Transfer Protocol Secure (HTTPS) to protect the transfer. HTTPS works on top of the Transport Layer Security (TLS) protocol, which is an end-to-end protocol used to establish encrypted tunnels between clients and origin servers. In the TLS handshake, clients and servers negotiate different parameters to establish such tunnels and, although it is not mandatory, the client should verify the identity of the server. Usually, the server sends its identification in the form

of a digital certificate, containing the server name, the Certificate Authority (CA) who has signed the certificate and the server's public encryption key. With this digital certificate, clients may validate the authenticity of the server, and may use the public encryption key of the server to negotiate the symmetric key, which will be used in the data transfer stage.

However, when a CDN is placed in the middle between clients and content providers, HTTPS does not provide all required security levels. As shown in Fig. **??**, in CDN deployments clients establish connections with replica servers provided by the CDN provider, and not with origin servers. In this scenario, several issues arise. Depending on how redirection is implemented by the CDN provider and content provider, the client may detect a name mismatch between the content provider's certificate and the replica server. For example, if the content provider replaces all existing URLs (URL rewriting) to the URLs used by the CDN provider, clients will use the CDN provider's certificate to establish TLS connections with the appropriate replica server. It is worth noting that the end user will notice a change in the accessed domain, which could create doubts about the authenticity of the content. On the other hand, URL rewriting requires content modifications, increasing the time required to publish content and reducing the flexibility to change CDN provider easily. This is the reason why the most used techniques for redirection are based on DNS resolution, where DNS servers redirect content provider names to CDN provider servers. Clients are not aware of this redirection, so this mechanism ends in a name mismatch at the client side when establishing the TLS connection.

In a CDN scenario, SCoT provides the mechanisms to authenticate the exchanged content to all parties involved in the deployed service. When content providers move content to the CDN provider (backend communication) using a SCoT connection, the origin server computes the corresponding credentials associated to the transferred content. In the content delivery stage (see **??**), the SCoT layer at the origin server transmits blocks of content and credentials associated to those blocks. The CDN replica server (see Fig. **??**) uses the received credentials to authenticate each block of content. When the whole content is received, and all blocks are authenticated, the CDN server stores both the content and its associated credentials. These credentials can be later re-utilized to authenticate the dissemination of content from the replica server to interested clients, and may be later removed when purging content from the CDN. This behavior is extremely important, because the CDN provider just needs to verify the authenticity of the content using the

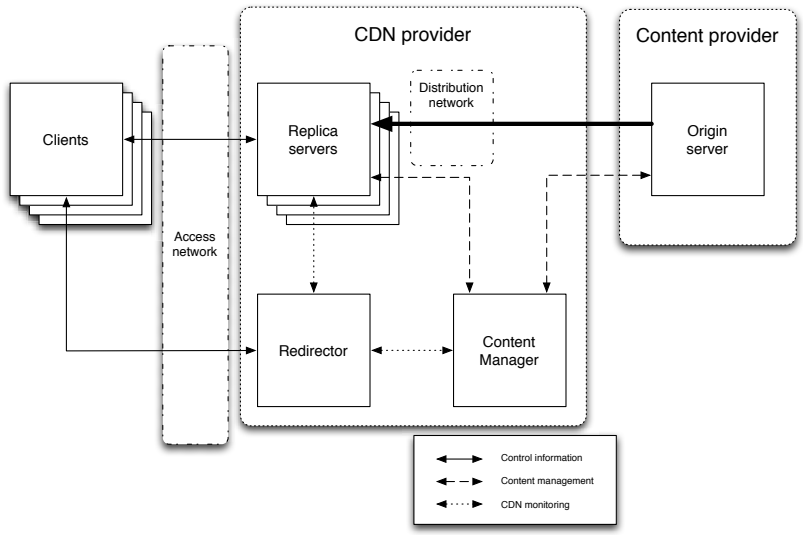

Figure 12: CDN architecture

(public) certificate provided by the content provider. With SCoT, it is the content provider (the origin server) the entity that generates all the credentials associated to the distributed content, so the private key is not disclosed to other entities.

We want to highlight that SCoT naturally supports the utilization of a pull approach by the CDN provider, where a replica server downloads content from the origin server as a consequence of a client request. In this case, although a solicited content and its credentials may not be available at the replica server upon receiving a request for that content over the SCoT connection maintained with the client, the server can use the SCoT protocol to retrieve the solicited content from the origin server. As content starts to be received through the SCoT connection established with the origin server, the replica server can obtain a reference to the security context activated by the origin server, and then use this reference to activate the same security context in the SCoT connection maintained with the client. This way, credentials received from the origin server will immediately be available to protect the relay of the content from the replica server to the client.

6.2. Software/Firmware/Plugins installation and upgrades

When users require a new service in their smart phones, tablets, computers or servers, they have a large amount of platforms to search for and install new software, firmware or plugins (for their web browsers, for example). The process of installing and updating software is critical, so the validity and integrity of the downloaded content have to be assured. Several security models are used to distribute software, like in **??**, where intermediate servers may cache the distributed content. In such scenarios, SCoT is a valid and viable solution to avoid vulnerabilities in software installation and upgrades without human intervention, which is the main drawback of the current mechanisms. As an additional advantage of SCoT in this type of applications, we want to mention that the protocol provides the mechanisms to authenticate the downloaded content while it is received, avoiding the consumption of resources (e.g., bandwidth, terminal battery, etc.) due to the transmission of subsequent parts of a corrupted or bogus content upon the unsuccessful authentication of one of its constituent blocks.

6.3. Internet of Things

In Internet of Things (IoT) communications, the vast majority of data generated by motes or small devices is produced to be consumed and processed by other devices or applications. Proposed IoT architectures use intermediate elements to process the information flow between IoT devices and applications (**?**), such as gateways, edge computing elements, data storages and data abstraction facilities. Additionally, users of the IoT platforms may collaborate and exchange data. Taking this into account, SCoT could be considered as a enabling protocol in IoT, to support the protected dissemination of information across the different elements of an IoT deployment.

6.4. Virtual Machines

In services like Cloud and NFV (Network Function Virtualization) physical infrastructures are shared among several users or tenants, mainly using the concept of virtual containers, where resources are isolated to prevent conflicts among those users. To efficiently control the available physical resources, an orchestrator decides where to deploy a given virtual container, as requested by an upper layer (i.e., the user of this service). Usually, the customer uploads a virtual container image to the service provider repository. After the proper server providing the necessary resources for the virtual

container is identified, the orchestrator copies the corresponding virtual container image from its repository to the selected host. In such large transfers, the block-by-block validation of the virtual container image and its supporting information is key before proceeding to start it at the server, to avoid potential security risks. The SCoT protocol can be used as an enabler to support the protected exchange of information in these virtual container environments.

In other services like Vagrant[6], LXD[7] or Docker[8], where there exists a virtual container repository where users upload and download their own virtual containers, the SCoT protocol can guarantee the authenticity of the exchanged images.

7. Conclusion

In this paper, we present a solution that allows content providers to protect the dissemination of information through third-party application-level infrastructures, such as content delivery networks or mirror sites. As a transport-layer protocol, SCoT may be utilized to deliver any application data to interested consumers, independently of the application-layer protocols used to request and serve the information (e.g., HTTP), being cryptographic operations executed transparently to application-layer entities. Our protocol offers a connection-oriented service and operates over TCP, which allows guaranteeing reliability and ordered delivery of information. A theoretical analysis, and the experiments done with an experimental prototype of the solution, show that SCoT may achieve performance results comparable to TCP, with an expected decrease on performance as the granularity of content authentication increases, due to the overhead introduced by the transmission of protocol-specific information and the larger number of cryptographic operations. Our future work will focus on developing SCoT as a standalone protocol, capable of operating directly over the IP layer, as well as on extensions to provide UDP-like transmission services to end-user applications. We will closely follow the work on ICN, and particularly on its innovative security solutions, to explore and evaluate synergies with the evolution of SCoT. In addition, we will study the application of the protocol to

[6]https://www.vagrantup.com/
[7]https://linuxcontainers.org/lxd
[8]https://www.docker.com/

specific use cases, particularly to aid secure content dissemination in IoT and Network Function Virtualization environments.

Acknowledgment

This article has been partially supported by the European H2020 5GinFIRE project (grant agreement 732497), and by the DRONEXT project (TEC2014-54335-C4-2-R) funded by the Spanish Ministry of Economy and Competitiveness. The work of Ignacio Soto has partially been supported by the Spanish Texeo project (TEC2016-80339-R) funded by the Spanish Ministry of Economy and Competitiveness.

References

Barnes, R., Iyengar, S., Sullivan, N., Rescorla, E., October 2017. Delegated credentials for tls. Tech. rep., IETF.
URL https://tools.ietf.org/html/draft-ietf-tls-subcerts-00

Bellissimo, A., Burgess, J., Fu, K., 2006. Secure software updates: Disappointments and new challenges. In: Workshop on Hot Topics in Security. USENIX, pp. 37–43.

Bittau, A., Boneh, D., Giffin, D., Hamburg, M., Handley, M., Mazieres, D., Slack, Q., Smith, E., October 2016. Cryptographic protection of tcp streams (tcpcrypt), expires may 4, 2017. Tech. rep., IETF.
URL https://datatracker.ietf.org/doc/draft-ietf-tcpinc-tcpcrypt

Cairns, K., Mattsson, J., Skog, R., Migault, D., October 2015. Session key interface (ski) for tls and dtls. Tech. rep., IETF.
URL https://tools.ietf.org/html/draft-cairns-tls-session-key-interface-01

Cisco Systems, 2014. The internet of things reference model. Tech. rep., Cisco Systems.

Dierks, T., Rescorla, E., Aug. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919.
URL http://www.ietf.org/rfc/rfc5246.txt

Erb, S., Salz, R., May 2016. A pfs-preserving protocol for lurk. Tech. rep., IETF.
URL https://tools.ietf.org/html/draft-erb-lurk-rsalg-01

Freier, A., Karlton, P., Kocher, P., Aug. 2011. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic).
URL http://www.ietf.org/rfc/rfc6101.txt

Gkantsidis, C., Karagiannis, T., VojnoviC, M., 2006. Planet scale software updates. In: Conference on Applications, Technologies, Architectures, and Protocols forComputer Communications. ACM, pp. 423–434.

Hallam-Baker, P., April 2016. Limited use of remote keys, protocol and reference. Tech. rep., IETF.
URL https://tools.ietf.org/html/draft-hallambaker-lurk-02

Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., Braynard, R. L., 2009. Networking named content. In: Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies. CoNEXT '09. ACM, New York, NY, USA, pp. 1–12.
URL http://doi.acm.org/10.1145/1658939.1658941

Kent, S., Seo, K., Dec. 2005. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), updated by RFCs 6040, 7619.
URL http://www.ietf.org/rfc/rfc4301.txt

Liang, J., Jiang, J., Duan, H., Li, K., Wan, T., Wu, J., 2014. When https meets cdn: A case of authentication in delegated service. In: Security and privacy (sp), 2014 IEEE symposium on. IEEE, pp. 67–82.

Stebila, D., Sullivan, N., Aug 2015. An analysis of tls handshake proxying. In: 2015 IEEE Trustcom/BigDataSE/ISPA. Vol. 1. pp. 279–286.

Thomson, M., Eriksson, G., Holmberg, C., October 2016a. An architecture for secure content delegation using http. Tech. rep., IETF.
URL https://tools.ietf.org/html/draft-thomson-http-scd-02

Thomson, M., Eriksson, G., Holmberg, C., October 2016b. Caching secure http content using blind caches. Tech. rep., IETF.
URL https://www.ietf.org/id/draft-thomson-http-bc-01.txt

Xylomenos, G., Ververidis, C. N., Siris, V. A., Fotiou, N., Tsilopoulos, C., Vasilakos, X., Katsaros, K. V., Polyzos, G. C., Second 2014. A survey of information-centric networking research. IEEE Communications Surveys Tutorials 16 (2), 1024–1049.