Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

J. Mattsson
J. Fornehed
G. Selander
F. Palombini
Ericsson

October 19, 2015

Controlling Actuators with CoAP
draft-mattsson-core-coap-actuators-00

Abstract

Being able to trust information from sensors and to securely control actuators is essential in a world of connected and networking things interacting with the physical world. In this memo we show that just using COAP with a security protocol like DTLS or OSCOAP is not enough. We describe several serious attacks any on-path attacker can do, and discuss tougher requirements and mechanisms to mitigate the attacks. While this document is focused on actuators, one of the attacks applies equally well to sensors using DTLS.

Table of Contents

1. Introduction

Being able to trust information from sensors and to securely control actuators is essential in a world of connected and networking things interacting with the physical world. One protocol used to interact with sensors and actuators is the Constrained Application Protocol (CoAP). Any Internet-of-Things (IoT) deployment valuing security and privacy would use a security protocol such as DTLS [RFC6347] or OSCOAP [I-D.selander-ace-object-security] to protect CoAP, but we show that this is not enough. We describe several serious attacks any on-path attacker (i.e. not only "trusted" intermediaries) can do, and discusses tougher requirements and mechanisms to mitigate the attacks. The request delay attack (valid for both DTLS and OSCOAP and described in Section 2.2) lets an attacker control an actuator at a much later time than the client anticipated. The response delay and mismatch attack (valid for DTLS and described in Section 2.3) lets an attacker respond to a client with a response meant for an older request. In Section 3, a new CoAP Option, the Repeat Option, mitigating the delay attack in specified.

2. Attacks

Internet-of-Things (IoT) deployments valuing security and privacy, MUST use a security protocol such as DTLS or OSCOAP to protect CoAP. This is especially true for deployments of actuators where attacks often (but not always) have serious consequences. The attacks

described in this section are made under the assumption that CoAP is already protected with a security protocol such as DTLS or OSCOAP, as an attacker otherwise can easily forge false requests and responses.

2.1. The Block Attack

An on-path attacker can block the delivery of any number of requests or responses. The attack can also be performed by an attacker jamming the lower layer radio protocol. This is true even if a security protocol like DTLS or OSCOAP is used. Encryption makes selective blocking of messages harder, but not impossible or even infeasible. With DTLS, proxies have access to the complete CoAP message, and with OSCOAP, the CoAP header and several CoAP options are not encrypted. In both security protocols, the IP-addresses, ports, and CoAP message lengths are available to all on-path attackers, which may be enough to determine the server, resource, and command. The block attack is illustrated in Figure 1 and 2.

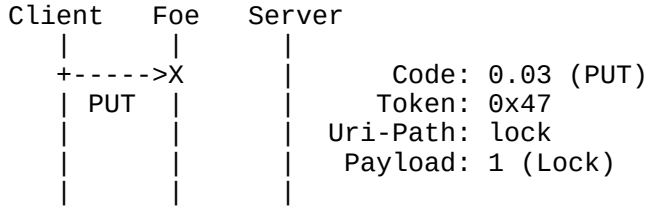

Figure 1: Blocking a Request

Where 'X' means the attacker is blocking delivery of the message.

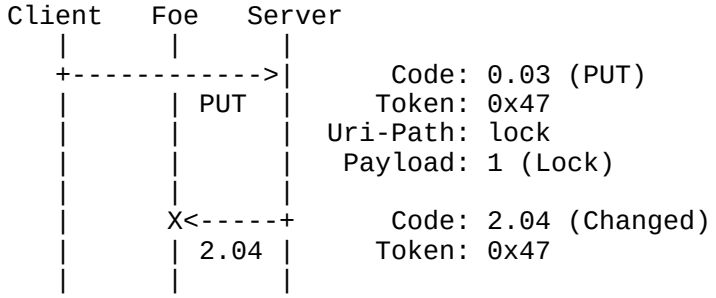

Figure 2: Blocking a Response

While blocking requests to, or responses from, a sensor is just a denial of service attack, blocking a request to, or a response from, an actuator results in the client losing information about the server's status. If the actuator e.g. is a lock (door, car, etc.), the attack results in the client not knowing (except by using out-of-

band information) whether the lock is unlocked or locked, just like the observer in the famous Schroedinger's cat thought experiment. Due to the nature of the attack, the client cannot distinguish the attack from connectivity problems, offline servers, or unexpected behavior from middle boxes such as NATs and firewalls.

Remedy: In actuator deployments where confirmation is important, the application MUST notify the user upon reception of the response, or warn the user when a response is not received. The application SHOULD also indicate to the user that the status of the actuator is now uncertain.

2.2. The Request Delay Attack

An on-path attacker may not only block packets, but can also delay the delivery of any packet (request or response) by a chosen amount of time. This is true even if DTLS or OSCOAP is used, as long as the delayed packet is delivered inside the replay window. The replay window has a default length of 64 in DTLS and is application dependent in OSCOAP. The attacker can control the replay window by blocking some or all other packets. By first delaying a request, and then later, after delivery, blocking the response to the request, the client is not made aware of the delayed delivery except by the missing response. The server has in general, no way of knowing that the request was delayed and will therefore happily process the request.

If some wireless low-level protocol is used, the attack can also be performed by the attacker simultaneously recording what the client transmits while at the same time jamming the server. The request delay attack is illustrated in Figure 3.

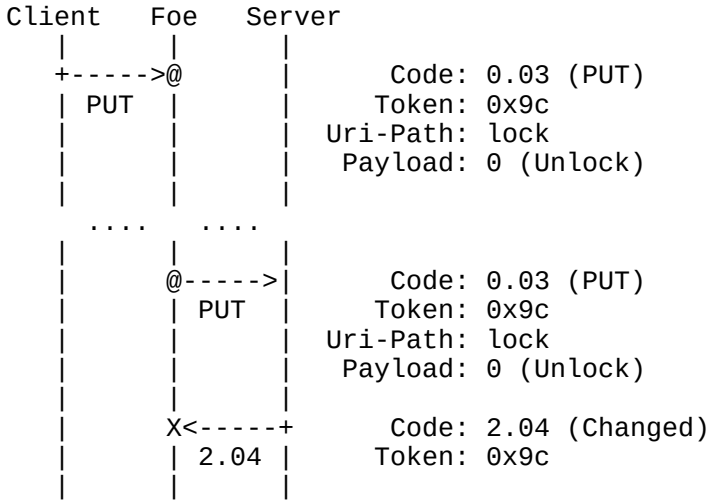

Figure 3: Delaying a Request

Where '@' means the attacker is storing and later forwarding the message (@ may alternatively be seen as a wormhole connecting two points in spacetime).

While an attacker delaying a request to a sensor is often not a security problem, an attacker delaying a request to an actuator performing an action is often a serious problem. A request to an actuator (for example a request to unlock a lock) is often only meant to be valid for a short time frame, and if the request does not reach the actuator during this short timeframe, the request should not be fulfilled. In the unlock example, if the client does not get any response and does not physically see the lock opening, the user is likely to walk away, calling the locksmith (or the IT-support).

If a non-zero replay window is used (the default in DTLS and unspecified in OSCOAP), the attacker can let the client interact with the actuator before delivering the delayed request to the server (illustrated in Figure 4). In the lock example, the attacker may store the first "unlock" request for later use. The client will likely resend the request with the same token. If DTLS is used, the resent packet will have a different sequence number and the attacker can forward it. If OSCOAP is used, resent packets will have the same sequence number and the attacker must block them all until the client sends a new message with a new sequence number (not shown in Figure 4). After a while when the client has locked the door again, the attacker can deliver the delayed "unlock" message to the door, a very serious attack.

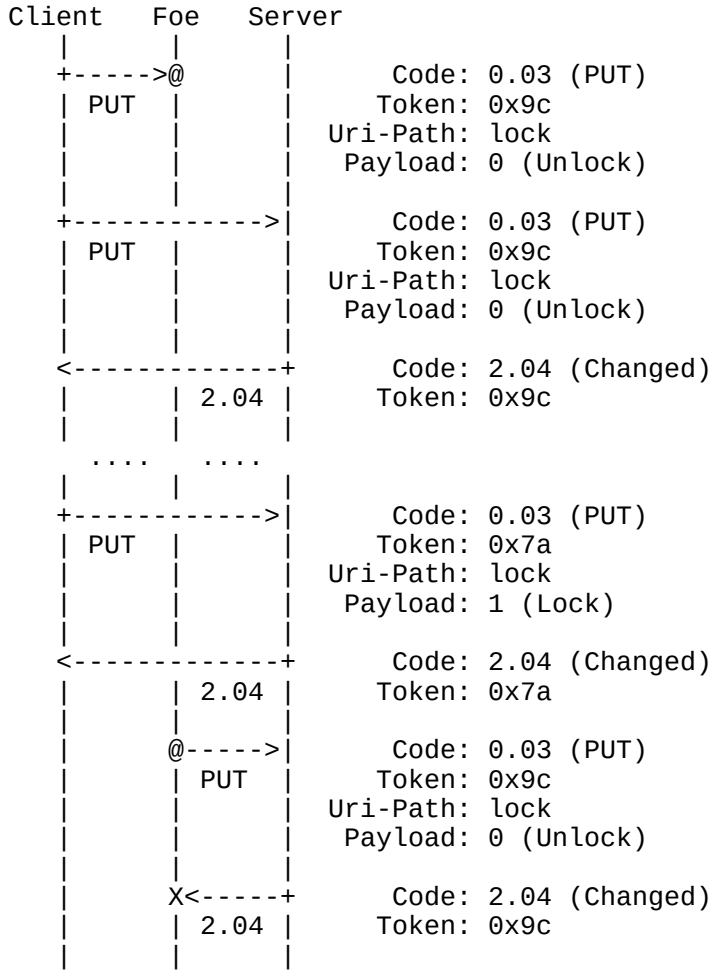

Figure 4: Delaying Request with Reordering

While the second attack (Figure 4) can be mitigated by using a replay window of length zero, the first attack (Figure 3) cannot. A solution must enable the server to verify that the request was received within a certain time frame after it was sent. This can be accomplished with either a challenge-response pattern or by exchanging timestamps. Security solutions based on timestamps require exactly synchronized time, and this is hard to control with complications such as time zones and daylight saving. Even if the clocks are synchronized at one point in time, they may easily get out-of-sync and an attacker may even be able to affect the client or the server time in various ways such as setting up a fake NTP server, broadcasting false time signals to radio controlled clocks, or expose

one of them to a strong gravity field. As soon as client falsely believes it is time synchronized with the server, delay attacks are possible. A challenge response mechanism is much more failure proof and easy to analyze. One such mechanism, the CoAP Repeat Option, is specified in Section 3.

Remedy: The CoAP Repeat Option specified in Section 3 SHALL be used for controlling actuators unless another application specific challenge-response or timestamp mechanism is used.

2.3. The Response Delay and Mismatch Attack

The following attack can be performed if CoAP is protected by a security protocol where the response is not bound to the request in any way except by the CoAP token. This would include most general security protocols, such as DTLS and IPsec, but not OSCOAP. The attacker performs the attack by delaying delivery of a response until the client sends a request with the same token. As long as the response is inside the replay window (which the attacker can make sure by blocking later responses), the response will be accepted by the client as a valid response to the later request. CoAP [RFC7252] does not give any guidelines for the use of token with DTLS, except that the tokens currently "in use" SHOULD (not SHALL) be unique.

The attack can be performed by an attacker on the wire, or an attacker simultaneously recording what the server transmits while at the same time jamming the client. The response delay and mismatch attack is illustrated in Figure 5.

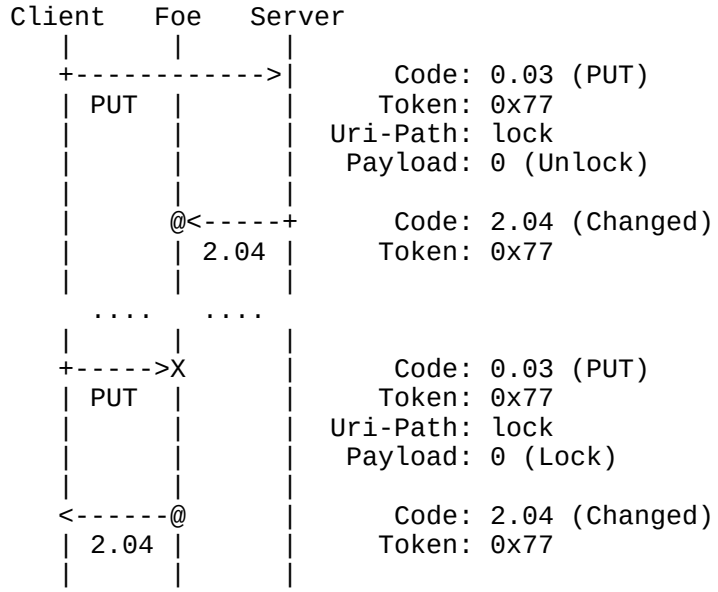

Figure 5: Delaying and Mismatching Response to PUT

If we once again take a lock as an example, the security consequences may be severe as the client receives a response message likely to be interpreted as confirmation of a locked door, while the received response message is in fact confirming an earlier unlock of the door. As the client is likely to leave the (believed to be locked) door unattended, the attacker may enter the home, enterprise, or car protected by the lock.

The same attack may be performed on sensors, also this with serious consequences. As illustrated in Figure 6, an attacker may convince the client that the lock is locked, when it in fact is not. The "Unlock" request may be also be sent by another client authorized to control the lock.

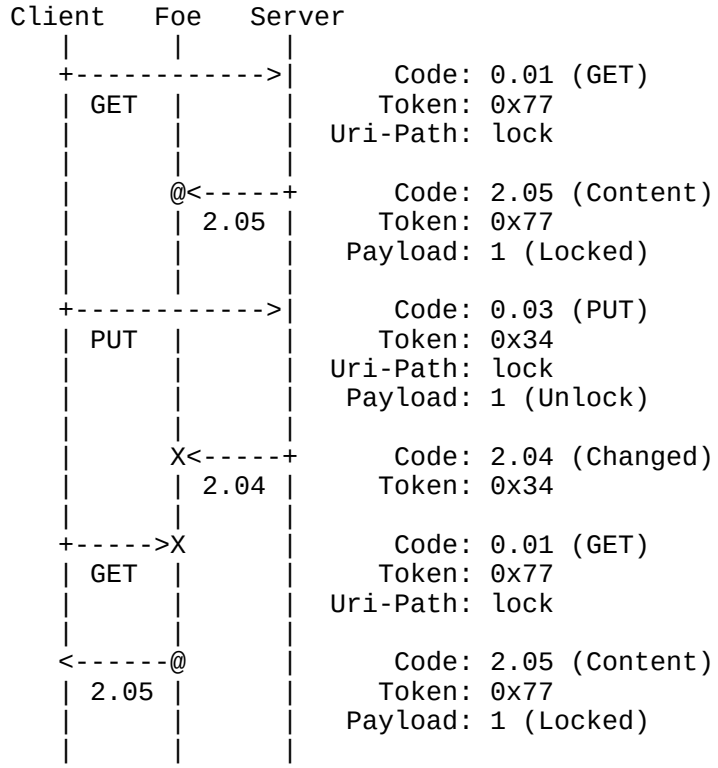

Figure 6: Delaying and Mismatching Response to GET

As illustrated in Figure 7, an attacker may even mix responses from different resources as long as the two resources share the same DTLS connection on some part of the path towards the client. This can happen if the resources are located behind a common gateway, or are served by the same CoAP proxy. An on-path attacker (not necessarily a DTLS endpoint such as a proxy) may e.g. deceive a client that the living room is on fire by responding with an earlier delayed response from the oven (temperatures in degree Celsius).

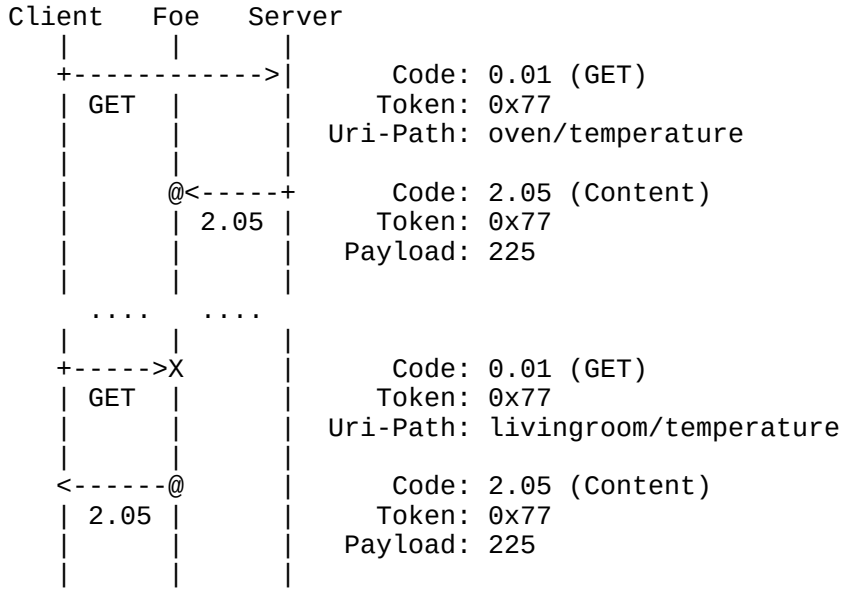

Figure 7: Delaying and Mismatching Response from other resource

OSCOAP is not susceptible to these attacks since it provides a secure binding between request and response messages.

Remedy: If CoAP is protected with a security protocol not providing bindings between requests and responses (e.g. DTLS) the client MUST NOT reuse any tokens for a given source/destination which the client has not received responses to. The easiest way to accomplish this is to implement the token as a counter and never reuse any tokens at all, this approach SHOULD be followed.

2.4. The Relay Attack

Yet another type of attack can be performed in deployments where actuator actions are triggered automatically based on proximity and without any user interaction, e.g. a car (the client) constantly polling for the car key (the server) and unlocking both doors and engine as soon as the car key responds. An attacker (or pair of attackers) may simply relay the CoAP messages out-of-band, using for examples some other radio technology. By doing this, the actuator (i.e. the car) believes that the client is close by and performs actions based on that false assumption. The attack is illustrated in Figure 8. In this example the car is using an application specific challenge-response mechanism transferred as CoAP payloads.

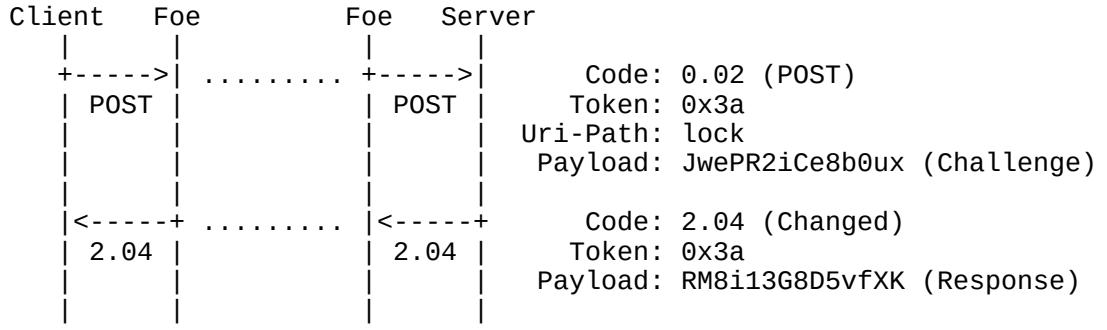

Figure 8: Relay Attack (the client is the actuator)

The consequences may be severe, and in the case of a car, lead to the attacker unlocking and driving away with the car, an attack that unfortunately is happening in practice.

Remedy: Getting a response over a short-range radio MUST NOT be taken as proof of proximity and therefore MUST NOT be used to take actions based on such proximity. Any automatically triggered mechanisms relying on proximity MUST use other stronger mechanisms to guarantee proximity. Mechanisms that MAY be used are: measuring the round-trip time and calculate the maximum possible distance based on the speed of light, or using radio with an extremely short range like NFC (centimeters instead of meters). Another option is to including geographical coordinates (from e.g. GPS) in the messages and calculate proximity based on these, but in this case the location measurements MUST be very precise and the system MUST make sure that an attacker cannot influence the location estimation, something that is very hard in practice.

3. The Repeat Option

The Repeat Option is a challenge-response mechanism for CoAP, binding a resent request to an earlier 4.03 forbidden response. The challenge (for the client) is simply to echo the Repeat Option value in a new request. The Repeat Option enables the server to verify the freshness of a request, thus mitigating the Delay Attack described in Section 2.2. An example message flow is illustrated in Figure 9.

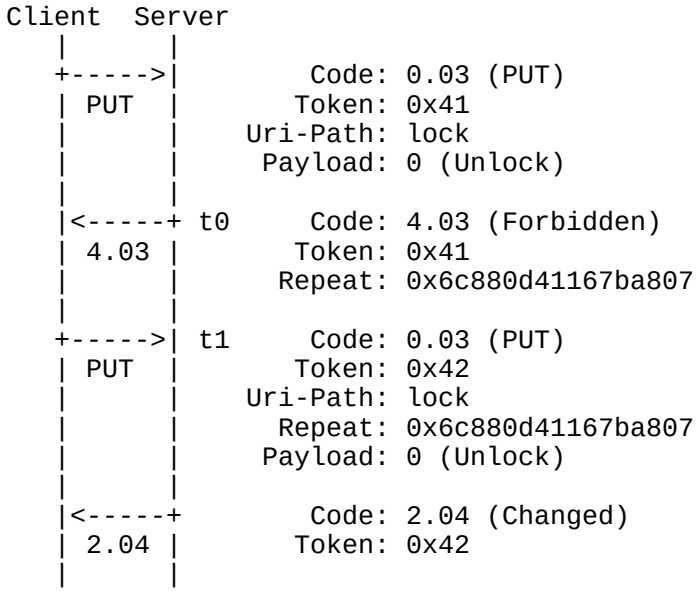

Figure 9: The Repeat Option

The Repeat Option may be used for all Methods and Response Codes. In responses, the value MUST be a (pseudo-)random bit string with a length of at least 64 bits. A new (pseudo-)random bit string MUST be generated for each response. In requests, the Repeat Option MUST echo the value from a previously received response.

The Repeat Option is critical, Safe-to-Forward, not part of the Cache-Key, and not repeatable.

Upon receiving a request without the Repeat Option to a resource with freshness requirements, the server sends a 4.03 Forbidden response with a Repeat Option and stores the option value and the response transmit time t0.

Upon receiving a 4.03 Forbidden response with the Repeat Option, the client SHOULD resend the request, echoing the Repeat Option value.

Upon receiving a request with the Repeat Option, the server verifies that the option value equals the previously sent value; otherwise the request is not processed further. The server calculates the round-trip time RTT = (t1 - t0), where t1 is the request receive time. The server MUST only accept requests with a round-trip time below a certain threshold T, i.e. RTT < T, otherwise the request is not processed further, and an error message MAY be sent. The threshold T is application specific.

An attacker able to control the server's clock with high precision, could still be able to perform a delay attack by moving the server's clock back in time, thus making the measured round-trip time smaller than the actual round-trip time. The times t0 and t1 MUST therefore be measured with a steady clock (one that cannot be adjusted).

EDITORS NOTE: The mechanism described above gives the server freshness guarantee independently of what the client does. The disadvantages are that the mechanism always takes two round-trips and that the server has to save the option value and the time t0. Other solutions involving time may be discussed:

- The server may simply send the client the current time in its timescale, i.e. a timestamp (option value = t0). The client may then use this timestamp to estimate the current time in the servers timescale when sending future requests (i.e. not echoing). This approach has the benefit of reducing round-trips and server state, but has the security problems discussed in Section 2.2.

- The server may instead of a pseudorandom value send an encrypted timestamp (option value = E(k, t0)). CTR-mode would from a security point be like sending (value = t0). ECB-mode or CCM-mode would work, but would expand the value length. With CCM, the server might also bind the option value to request (value = AEAD(k, t0, parts of request)). This approach does not reduce the number of round-trips but eliminates server state.

4. IANA Considerations

This document defines the following Option Number, whose value have been assigned to the CoAP Option Numbers Registry defined by [RFC7252].

Number	Name
29	Repeat

5. Security Considerations

The whole document can be seen as security considerations for CoAP.

6. References

6.1. Normative References

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

6.2. Informative References

[I-D.selander-ace-object-security] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "June 29, 2015", draft-selander-ace-object-security-02 (work in progress), June 2015.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

Authors' Addresses

John Mattsson
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.mattsson@ericsson.com

John Fornehed
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.fornehed@ericsson.com

Goran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com