# Lightweight and adaptable solution for security agility

**Valter Vasić[1], Miljenko Mikuc[1] and Marin Vuković[1]**
[1] Faculty of Electrical Engineering and Computing
Unska 3, Zagreb, Croatia
[e-mail: valter.vasic@fer.hr, miljenko.mikuc@fer.hr, marin.vukovic@fer.hr]
*Corresponding author: Valter Vasić

*Received August 25, 2015; revised November 17, 2015; accepted February 3, 2016;*
*published March 31, 2016*

## Abstract

Secure communication is an important aspect of today's interconnected environments and it can be achieved by the use of cryptographic algorithms and protocols. However, many existing cryptographic mechanisms are tightly integrated into communication protocols. Issues emerge when security vulnerabilities are discovered in cryptographic mechanisms because their replacement would eventually require replacing deployed protocols. The concept of cryptographic agility is the solution to these issues because it allows dynamic switching of cryptographic algorithms and keys prior to and during the communication. Most of today's secure protocols implement cryptographic agility (IPsec, SSL/TLS, SSH), but cryptographic agility mechanisms cannot be used in a standalone manner. In order to deal with the aforementioned limitations, we propose a lightweight cryptographically agile agreement model, which is formally verified. We also present a solution in the Agile Cryptographic Agreement Protocol (ACAP) that can be adapted on various network layers, architectures and devices. The proposed solution is able to provide existing and new communication protocols with secure communication prerequisites in a straightforward way without adding substantial communication overhead. Furthermore, it can be used between previously unknown parties in an opportunistic environment. The proposed model is formally verified, followed by a comprehensive discussion about security considerations. A prototype implementation of the proposed model is demonstrated and evaluated.

***Keywords:*** cryptographic agility, key exchange, algorithm agreement, communication model, model verification

## 1. Introduction

Securing data and communication channels implies the use of cryptography. To achieve primary security requirements (confidentiality, integrity and availability) different types of cryptographic algorithms are used. Cryptographic algorithms can be divided into three groups: symmetric encryption, asymmetric encryption and cryptographic hashes. These algorithms are often used in different combinations to achieve better performance and various security requirements. For example, digital signature combines cryptographic hashes and asymmetric encryption, whereas authenticated encryption combines symmetric cryptography and cryptographic hashes.

There is a vast amount of cryptographic algorithms currently available. They are all constantly being improved and at the same time new vulnerabilities are being discovered and exposed. An algorithm with vulnerabilities, from the crypto analysis point of view, can still be safe for usage, but the identified vulnerability points to a weakness in its definition [1] [2] [3]. Such a weakness can eventually be used for various attacks on systems that are secured by that algorithm, so vulnerable algorithms should be replaced as soon as possible [4] [5] [6].

However, replacing cryptographic algorithms that are tightly integrated into communication protocols would eventually require changing the protocols themselves. This usually means that the running software or firmware needs to be updated. A good overview of the challenges and a solution for issuing firmware updates is covered in [7]. Changing protocols may prove to be an issue when dealing with typically large and complex communication systems. This is the reason why cryptographic algorithms should never be tightly integrated into a communication protocol and the solution is to use the concept of cryptographic agility [8]. Cryptographic agility is characterized by the ability to change cryptographic algorithms and keys while using the same protocols and deployed systems. On the other hand, using agility makes it possible to use the same cryptographic algorithms or protocols over a wide array of communication protocols thus enabling easier replacement of potentially insecure cryptographic algorithms.

Cryptographic agility [9] is a known principle that is already integrated into most widely deployed secure channel protocols, such as IPsec [10], SSL [11], TLS [12], SSH [13]. However, each of the implemented agility mechanisms is strictly tied to that solution and cannot be easily extracted as a standalone mechanism applicable for other purposes. The possible solution that would enable this functionality is an adaptable cryptographically agile protocol that we propose in this paper. This solution could be used on various network layers, architectures and devices. Furthermore, the aim would be the agreement of cryptographic prerequisites (algorithms and keys) in order to enable various security requirements in an interconnected environment. Using the proposed cryptographically agile solution enables protection of protocols that don't have any security mechanisms deployed and enable development of new secure protocols on top of the agreed primitives.

Our lightweight and adaptable approach is easily applicable in currently emerging areas, such as Internet of Things (IoT). Security isn't always integrated in current IoT solutions and is typically planned at a later stage as an upgrade. The communication between IoT devices needs to be secure which requires an agreement protocol to enable communicating parties to agree on a cryptographic algorithm and keys used to protect the exchanged messages. The IoT ecosystem needs a flexible and adaptable agreement mechanisms because IoT devices have limited computing and bandwidth resources. Secure communication is possible only after communicating devices agree on a set of cryptographic algorithms and keys. Since IoT services are driven by the underlying data sources, it is critical to validate a sensor as a credible data source and to protect the exchanged data.

Secure channel protocols have many components and thus tend to be fairly complex (e.g. SSL/TLS, IPsec, SSH). That complexity makes it hard, or even impossible in certain cases, to create a formal model that can be thoroughly verified to achieve provable security. The lack of security proofs is probably one of the causes of the recent increase of attacks on SSL/TLS as stated in [14]. Verification of certain solutions can also reveal previously unknown vulnerabilities as shown in [15].

Our motivation was to achieve provable security by extracting and integrating key exchange with algorithm agreement into a standalone solution. As a result, we designed a lightweight solution that is formally verified and achieves distribution of keys and algorithms in a provably secure manner. The conducted formal verification proves resilience to (pre)replay attacks [16] and man-in-the-middle attacks. The presented cryptographic agility solution enables lightweight and simple implementation that can be used on different network layers. Exchange of cryptographic keys and algorithms is preformed using only four messages within two round trips. It is designed to have a small amount of states to avoid implementation issues. It features a number of security mechanisms that are fundamental parts of the specification.

In paper [17] we addressed hash agility in the SEND [18] protocol. Since the hash agility presents just one topic in the area of cryptographic agility, we focused on general cryptographic algorithm agility in [19]. Our idea is further improved in this paper by introducing key exchange and thus presents a complete security agility solution that encompasses a set of cryptographic algorithms with the required key types.

The paper is organized as follows. In Section 2 we introduce the modeled protocol with message and communication description. In Section 3 we discuss the security aspects of the protocol. Section 4 analyses the formal security model specification and verification, while Section 5 evaluates our prototype protocol implementation and shows how the protocol behaves in a realistic environment. In Section 6 we give an overview of existing protocols with their strengths and weaknesses. Section 7 provides the direction of our future work and we conclude in Section 8.

## 2. Solution description

The goal of the proposed Agile Cryptographic Agreement Protocol (ACAP) is to give all secure communication prerequisites to the communicating parties. ACAP, after a successful exchange, provides the communicating parties with the following prerequisites:

- A shared secret established through Diffie Hellman that can be used to provide perfect forward secrecy. [20]
- Public keys used for authentication.
- List of cryptographic algorithms supported by both communicating parties, that will later be used to ensure the needed security requirements.

ACAP is based on the sign-and-mac (SIGMA) principle presented by Krawczyk in [21]. SIGMA combines the Diffie-Hellman (DH) [22] key exchange, digital signature and hashed message authentication code (HMAC) [23] [24] in a way that prevents the attacker to mount a man-in-the-middle attack on the protocol (explained in Section 3). The JFK protocol [25] also uses the sign-and-mac approach, which results in similar message definitions for the first two messages. Algorithm agreement during the message exchange is similar to the one defined for the SSH transport layer protocol [26].

## 2.1 Notation

The notation used for protocol description is shown below, where the variable X/x represents either the initiator (I/i) or the responder (R/r):

| | |
|---|---|
| $g^X$ | DH key exchange exponential and group info [22] |
| $N_X$ | nonce (unique temporary value) used for one agreement, represents a unique session ID that is used for the exchange |
| K | session key is the result of a key derivation function on the DH shared secret (derived from $g^i$, r and $g^r$, i) and both nonces. $K = KDF(g^{ir}, N_I, N_R)$ |
| $PK_X$ | public key or certificate |
| $S_X\{\}$ | digital signature using a secret key, that can be verified with the corresponding public key |
| $H_K\{\}$ | HMAC using session key K |
| $L_X$ | list of supported cryptographic algorithms |
| $E_X$ | error message that contains information about the error that occurred |

## 2.2 Messages

ACAP consists of four messages that are needed for the exchange of public keys ($PK_X$), agreement on a shared secret used for generating the session key K and finally agreement on cryptographic algorithms that will be used for further secure communication. The messages are defined as follows:

$$
\begin{aligned}
&INIT_I(I \rightarrow R): && g^i, N_I \\
&INIT_R(R \rightarrow I): && g^r, N_R, PK_R, S_R\{g^r\}, H_K\{PK_R, N_I, N_R\} \\
&LIST_I(I \rightarrow R): && PK_I, L_I, S_I\{L_I, g^i, g^r\}, H_K\{PK_I, N_R, N_I\} \\
&LIST_R(R \rightarrow I): && L_R, S_R\{g^i, g^r, N_I, N_R, L_R\}
\end{aligned}
$$

The first two messages contain the Diffie-Hellman exponentials for both the initiator and responder that are needed for the derivation of a session key K that will be used to calculate HMAC protection for messages. The second and third message are used to distribute public keys of the communicating parties, which are used to protect the message exchange using digital signatures. Even though using both HMAC and digital signatures seems redundant, this is not the case, and represents the base of a sign-and-mac (SIGMA) exchange [21]. The key idea behind SIGMA is to guarantee that the session key K is shared with the communicating party that owns the private key that corresponds to either the initiator ($PK_I$) or responder ($PK_R$). This is achieved by digitally signing the Diffie-Hellman exponentials and applying HMAC to the public key that was used to calculate the digital signature. This is to prevent binding of the session key K to a wrong public key and preventing a man-in-the-middle attack which is described in Section 3 and formally verified as described in Section 4.

## 2.3 Message exchange

A regular ACAP exchange is shown in **Fig. 1**. It includes the calculation and distribution of all parameters.

The $INIT_I$ message contains the first DH exponential ($g^i$) together with the DH group info and the initiators nonce ($N_I$). If the group is supported the responder answers to the challenge with the $INIT_R$ message.

Before sending the $INIT_R$ message the responder needs to calculate the session key K, from the shared secret established through DH, using a key derivation function on both nonces and the shared secret.

$INIT_R$ contains the second DH exponential ($g^r$) together with the responder public key ($PK_R$) and nonce ($N_R$). This message is secured by the digital signature ($S_R\{g^r\}$) of the responder's Diffie-Hellman exponential coupled with a HMAC ($H_K\{PK_I,N_R,N_I\}$) of the responder public key and both nonces using the session key K established through Diffie-Hellman.

Upon receipt of the $INIT_R$ message the initiator calculates the session key K in the same way as the responder. This is needed to produce the message $LIST_I$, which contains the initiators public key ($PK_I$) together with the initiators list of supported cryptographic algorithms ($L_I$). The digital signature ($S_I\{L_I,g^i,g^r\}$) protects the list, whereas the initiator's public key is protected by a HMAC ($H_K\{PK_I,N_R,N_I\}$) using the DH session key K. $LIST_R$ is used to transfer the responders list of supported algorithms ($L_R$), together with a digital signature ($S_R\{g^i,g^r,N_I,N_R,L_R\}$) that protects the list.

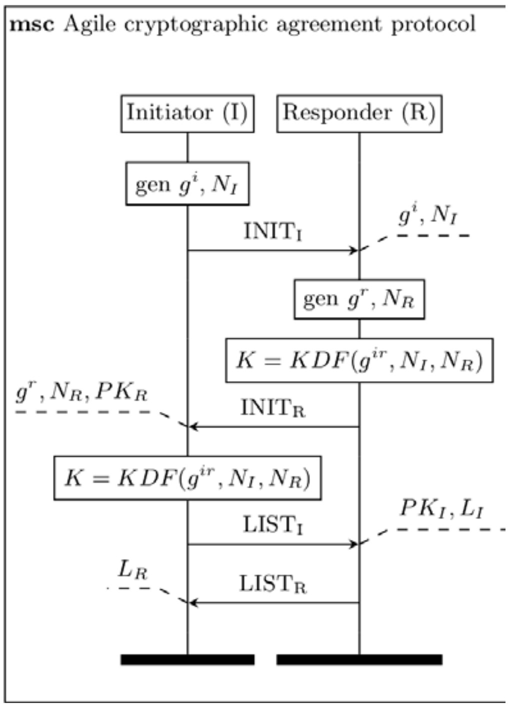

**Fig. 1.** ACAP message exchange diagram

## 2.4 Exception handling

If any verification of the sent messages fails, an abort message is sent to the other party.

These messages can be generated in the following scenarios: incompatible DH groups, digital signature or HMAC verification failure, no cryptographic algorithms in common between the initiator and the responder. They are different for the initiator and the responder and are defined as follows:

$$\mathrm{ABORT}_{\mathrm{R}}(R \rightarrow I): \quad \mathrm{E}_{\mathrm{R}},\mathrm{N}_{\mathrm{I}},\mathrm{PK}_{\mathrm{R}},\mathrm{S}_{\mathrm{R}}\{\mathrm{N}_{\mathrm{I}},\mathrm{E}_{\mathrm{R}}\}$$
$$\mathrm{ABORT}_{\mathrm{I}}(I \rightarrow R): \quad \mathrm{E}_{\mathrm{I}},\mathrm{N}_{\mathrm{R}},\mathrm{PK}_{\mathrm{I}},\mathrm{S}_{\mathrm{I}}\{\mathrm{N}_{\mathrm{I}},\mathrm{E}_{\mathrm{I}}\}$$

The ABORT messages contain the error message, nonce and public key and a digital signature of the nonce and error message. The digital signature is needed to avoid forgery of false ABORT messages that could enable possible attacks.

## 2.5 Cryptographic algorithm agreement

List of supported cryptographic suites and algorithms is organized according to their purpose. Sample lists and the agreement result with multiple algorithm categories are shown in **Fig. 2**.

```
Algorithm list client:
{
 "hash":[
   "SHA-256",
   "RIPEMD",
   "SHA-1"],
 "secret_key":[
   "AES-CTR_256",
   "3DES_192",
   "AES-CBC_128"],
 "public_key":[
   "RSA_1024",
   "RSA_2048",
   "ECDSA_192"]
}
```

```
Algorithm list server:
{
 "hash":[
   "SHA3-512",
   "SHA-512",
   "SHA-256"],
 "secret_key":[
   "AES-CTR_256",
   "Salsa20_256",
   "AES-CBC_128"],
 "public_key":[
   "ECDSA_192",
   "ECDSA_224",
   "RSA_2048"]
}
```

```
Agreed algorithms:
{
 "hash":
   "SHA-256",


 "secret_key":
   "AES-CTR_256",


 "public_key":
   "RSA_2048"


}
```

**Fig. 2.** Example algorithm lists and agreement result

For cryptographic algorithm agreement we adopt the principles from the SSH transport layer protocol [26]. For each algorithm type in the list we choose the first algorithm in the client list that is also in the server list. The result of the agreement procedure is shown in **Fig. 2**. This example uses three different algorithm types (hash, secret key algorithms and public key algorithms), but additional cryptographic algorithms (e.g. Diffie-Hellman algorithms, key derivation functions, etc.) could be added to the negotiation if needed.

All algorithms should be ordered by their respective strength and key length. Alternatively, they can be ordered by their preference in regards to computing speed and efficiency. A list of cryptographic algorithm and key length advisories can be found on the BlueKrypt site[1].

ACAP does not handle renegotiation itself. Once the algorithms and keys are agreed between parties, the application using ACAP should decide the duration of the negotiated parameters. The suggested re-keying and agreement interval is 1 hour, the usual default for IPsec Security Associations[2]. This parameter can be tuned based on the environment where the protocol is deployed.

[1] Keylength – Cryptographic Key Length Recommendation – http://www.keylength.com/en/
[2] https://wiki.strongswan.org/projects/strongswan/wiki/ExpiryRekey

## 3. Discussion and security considerations

Our aim is to provide a lightweight protocol to simplify its integration and formal verification. As a result, we have minimized potential attack points on the protocol. The amount of features introduced in a communication protocol is directly correlated to the possible weaknesses in the protocol and in the protocol implementation. Therefore, we removed some features like renegotiation.

The messages include only the data needed to protect the negotiation itself. Although nonces and DH exponentials are not contained in the messages itself they are present in the HMACs and digital signatures to prevent replay attacks. The sign-and-mac principle [21] is used to prevent man in the middle attacks because it implicitly ties the session key K (calculated from $g^i$ and $g^r$ DH values) to the public keys $PK_I$ and $PK_R$ that belong to the communicating parties.

*Partial absence of agility in the main exchange* - The protocol exchange consists of only four messages and the entire negotiation process is always conducted with fresh nonce values. This makes an attack on the fixed versions of cryptographic algorithms used in ACAP infeasible. Even if a vulnerability was found in currently used algorithms it could hardly be in a way to threaten a running exchange. Therefore, only the Diffie-Hellman group info is exchanged and that ensures shared secrets are generated by using the best currently supported DH parameters and that the resulting secret has a sufficient amount of security bits. The first version of ACAP should be deployed by using the latest widely supported versions of cryptographic algorithms, such as SHA-3 hash for HMAC and elliptic curve cryptography for asymmetric algorithms (DH and digital signatures). It must also be noted that a trivial introduction of negotiating algorithms used for the ACAP exchange would require at least one more message and introduce more failure points that would enable the attacker to lower the security of the exchange to the weakest cryptographic algorithms that are currently supported.

*Computational independence of secret keys* - a key part of sign-and-mac approach security lies in the need for computational independence of secret keys that are used to conduct the exchanges. Even though the keys are derived from the same shared secret they must be created in an independent way, as described in the appendix of [21]. After the ACAP exchange is conducted it outputs the algorithms that need to be used together with a secret key that is derived in a computationally independent way from the shared secret achieved through Diffie-Hellman. The shared secret is stored only in the running memory and deleted immediately after the exchange to prevent data leakage.

*Denial of service mitigation* - To create $INIT_R$ a responder should conduct the expensive operation of generating a safe DH prime coupled with calculating the session key K, signing the sent data and calculating HMAC. This is the most resource expensive operation for the responder and presents a potential attack point. The attacker could create a large amount of spoofed $INIT_I$ messages, which would cause resource depletion on the responder side due to responding $INIT_R$ messages. We mitigate this by using the same DH exponential for a short period of time and precomputing the more expensive part of the $INIT_R$ message ($S_R\{g^r\}$). Thus the only operations a server needs to conduct upon receipt, is the calculation of the session key and generation of HMAC for the last part of the message ($H_K\{PK_R,N_I,N_R\}$), while everything else is precomputed.

*MITM prevention* - In a man in the middle attack scenario, the attacker presents himself as the initiator to the responder and vice versa. This means that the negotiated secret keys and algorithms will be bound to the public keys with which they were negotiated. If the attacker changes the $INIT_I$ message ($g^i$, $N_I$), and provides a different Diffie-Hellman exponential (e.g. $g^{i'}$), it could also generate the returning $INIT_R$ message by using a different set of keys and a fresh DH exponential ($g^{r}$', $N_R$, $PK_R$', $S_R$'{$g^{r}$'}, $H_K$'{$PK_R$',$N_I$,$N_R$}). This would also lead to the changing of the session key K into K'. Subsequently all communication between the initiator and responder ($LIST_I$ and $LIST_R$ messages) would fail on the HMAC/digital signature verification and the communicating parties could not bind any negotiated results and shared secrets to their respective identities (public keys). Resilience to man-in-the-middle attacks has also been proven by the formal verification that we conducted on the security model of our protocol.

*Late replay prevention* - The aim of the attacker can be to capture an agreement and replay the same agreement at a later point in time, so that the communicating sides would use old keys and algorithms. If the attacker sent the captured $INIT_I$ message to the responder, the responder would generate a new nonce $N_R$. This nonce should be used in the HMAC ($H_K${$PK_I$,$N_R$,$N_I$}) contained in the $LIST_I$ message, which can't be done in a passive replay scenario. An active replay scenario would involve the possibility of the attacker to change replayed messages. This is countered by using a different DH exponential for later exchanges. This means that all further HMAC verifications would fail.

*Trust establishment* - ACAP enables the possibility of exchanging signed certificates instead of just public keys in the $INIT_R$ and $LIST_I$ messages. Thus the application using our protocol could manage certificate verification to ensure trust between communicating parties. This provides the same level of trust as in SSL/TLS and IPsec by introducing the same independent systems that are used for guaranteeing trust.

## 4. Formal model verification

The SIGMA approach that we use in ACAP is formally verified in [27]. We needed to formally model and verify ACAP, due to the changes we made in regard to SIGMA. The modeling and verification was done using Scyther [28], an automatic verification tool for security protocols.

Security requirements were defined in the security protocol description language (SPDL) used by Scyther [29] [30]. The security of ACAP is covered by three main requirements: 1) immutability of the DH exponentials, 2) secrecy of the shared secret used to derive session keys and 3) the inability of the attacker to interfere with the message exchange. From these requirements we constructed the following Scyther claims:

1. Running($g^{i}$,$g^{r}$) - The DH exponentials ($g^{i}$,$g^{r}$) must not be changed during the message exchange. This guarantees that both sides will use the same shared secret to generate secret keys.
2. Secret(K) - The shared secret (i.e. session key K) must remain secret throughout the whole exchange. Only then we can rely on that secret to be the input into a key derivation function for generating session keys.
3. Niagree(I, R) - Non-injective agreement for both communicating parties. This means that all messages have been sent and received without the possibility of an attacker interfering, consequently proving resilience to (pre-)replay and man-in-the-middle attacks.

Additional security claims were specified and verified for both parties to ensure the security of the proposed protocol: aliveness (Alive(I, R)), weak agreement (Weakagree(I, R)) and non-injective synchronization (Nisynch(I, R)). These claims are satisfied if the non-injective agreement claim holds, because their definition is contained in the non-injective agreement specification [31]. The hierarchy of security properties is shown in [30].

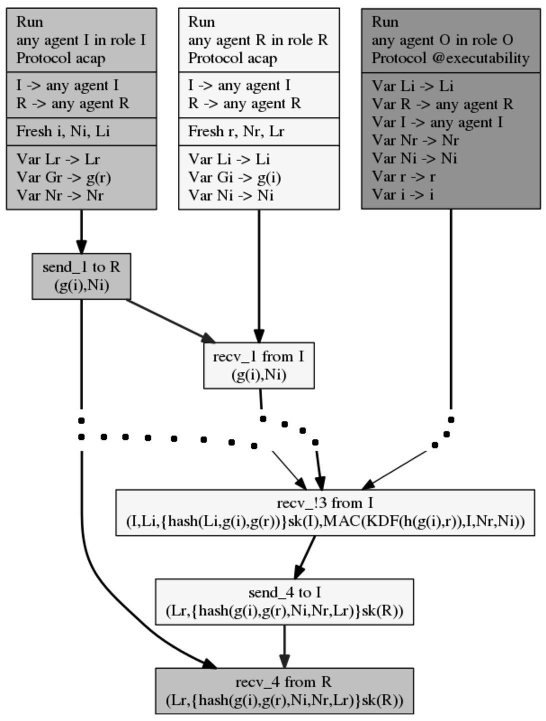

**Fig. 3.** Scyther characterization outline

Models in SPDL are defined by using protocols and roles. Each protocol can contain two or more roles that define multiple send and receive events. The modeling of protocol was done by defining ACAP with the initiator and responder roles in the main part. Additionally, to enable the specification of Diffie-Hellman primitives we used a helper (executability) protocol since this currently cannot be done in Scyther. [29]

A simplified outline of the Scyther characterization, by using three parallel runs, is shown in **Fig. 3**. The figure shows two main roles (initiator I and responder R) along with the role in the executability protocol O, which is necessary to define that the result of DH calculation is the same on the initiator and responder side ($((g^i)^r==(g^r)^i)$). The outline also shows the send (i.e. `send_X`) and receive (i.e. `recv_X`) events which are used to define the message exchange in Scyther. There is a total of two send and receive events for each of three roles used for verification. For further information on the verified model please refer to the full ACAP specification available on the link presented at the end of this section.

Scyther assumes that all roles have access to all public keys. However, in ACAP the public keys need to be exchanged between the initiator and responder. Therefore, to model this behavior the public key has been abstracted by the corresponding role. Such a role then presents the identity to the same extent as a public key.

The verification was conducted in an unbounded way without limiting the number of protocol runs and has shown that all defined requirements (Scyther claims) are fulfilled. The final verified version of the model is available on following link: http://public.tel.fer.hr/acap

## 5. Solution evaluation

To test and evaluate the proposed protocol and to show the adaptability of our design we have implemented a prototype in Python. The prototype implementation can be used to negotiate secure communication prerequisites on various network layers: Ethernet, IP, TCP and UDP. The protocol logic is shared for all communication layers which can be seen in the prototype code published on the same page as the formal verification model.

All message parts are currently delimited by using a delimiter but this can easily be migrated to a format with prepended lengths. The delimiter is also protected by the digital signatures to avoid trivial attacks on the implementation. Message parts need to be delimited in some way because almost all of them are of variable length: DH exponential lengths depend on the DH group used, public key lengths can also be of different sizes based on the standard used, digital signatures are the same size as the public keys used, HMAC lengths are different based on the hash algorithms used and the algorithm list length also varies based on the supported cryptographic algorithms.

We have tested our implementation by using the IMUNES[3] network emulation tool on Linux[4]. The first test was done by conducting the protocol exchange with various delay values on the link between the initiator and responder. Total agreement time was measured on the initiator side and is shown in **Fig. 4**. The delay in both ways was increased from 0ms up to 400ms (maximum round trip time was 800ms) and for every value a minimum of 30 measurements was done. We performed the tests for all transport protocols (Ethernet, IP, UDP and TCP) with the use of standard Diffie-Hellman and RSA and with the use of elliptic curve Diffie-Hellman and ECDSA. Measurements for connectionless protocols (Ethernet, IP, UDP) did not differ from each other, while TCP agreement times demonstrate a steeper line because of the initial three-way handshake. Therefore, in **Fig. 4** we have included only TCP and UDP agreement times when using standard and elliptic curve algorithms. The figure shows that using elliptic curve algorithms gives overall slightly better performance than using standard algorithms. For standard DH we used a 1024-bit mod group while for ECDH we used a 160-bit curve which provide similar levels of security. We also notice that network delay has a much greater impact than the usage of different cryptographic algorithms for the machine on which our measurements were performed.

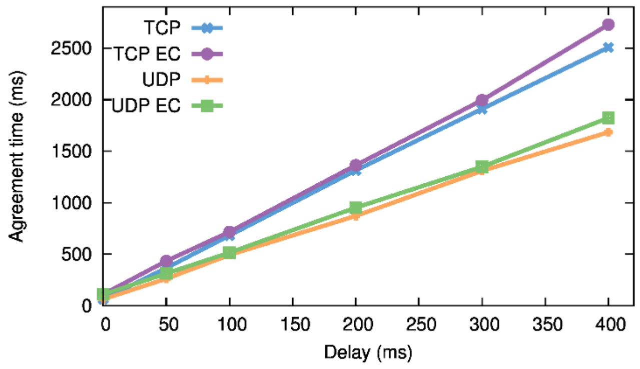

**Fig. 4.** Agreement times for the initiator side

A second test was conducted to measure message sizes when using different RSA public key sizes (768, 1024, 1536, 2048, 3072 and 4096 bits). Results are shown in **Fig. 5**. $INIT_I$ messages don't depend on public key size whereas $LIST_R$ has just one signature and scales accordingly. $INIT_I$ messages don't depend on public key size whereas $LIST_R$ has just one signature and scales accordingly.

[3] Integrated Multiprotocol Network Emulator / Simulator - http://imunes.net
[4] The tests were performed on a physical machine with a Core i7 processor and 8GB of RAM.

Fig. 5 also includes a line that represents the standard Ethernet MTU size of 1500 bytes. When using 4096 bit RSA keys the MTU limit is crossed. This can be countered by introducing elliptic curve public key algorithms, which are much smaller than the RSA keys used for the same level of security.

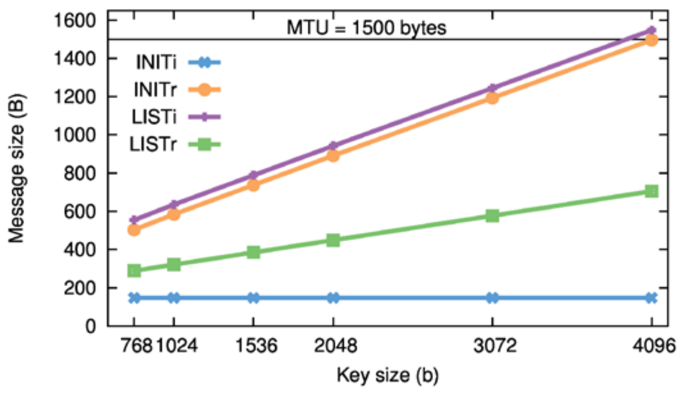

Fig. 5. Message size in respect to public key size

Furthermore, we measured initiator and responder message creation and processing times and average values of 30 agreements over UDP are given in Table 1. The table shows that the initiator side has an overall higher message creation and processing time than the responder. The INIT$_I$ message on the initiator side takes significantly more time because the initiator has to generate a strong Diffie-Hellman prime. Although the responder side also has to generate a similar prime, this is done as a background job every 30 seconds to avoid resource depletion and mitigate denial of service attacks. This difference is also present in the total times, which also include the delay caused by network communication and processing times for the other side of the communication.

Table 1. Message creation and processing times for both sides

| | Initiator | Responder |
|---|---|---|
| INIT$_I$ | 40.847 ms | 4.087 ms |
| INIT$_R$ | 4.298 ms | 0.155 ms |
| LIST$_I$ | 0.358 ms | 0.299 ms |
| LIST$_R$ | 0.188 ms | 0.290 ms |
| Total | 54.128 ms | 10.785 ms |

Our prototype implementation is available on the same web page as the verification model and verification results.

## 5.1 Usage on different layers

In order to implement and deploy ACAP on different layers, certain mechanisms are required. These mechanisms are divided into two main groups: 1) reliable transfer (TCP) and 2) unreliable transfer (UDP, IP, Ethernet) mechanisms.

ACAP is a message based protocol. When using stream based reliable transfer (TCP), ACAP messages have to be delimited. This is done by prepending the message length to the message.

Unreliable transfer protocols are message based, but we need to detect message loss that might occur during communication and resend the message if necessary. This issue is solved in the same manner as in DTLS [32] by using the expire timer: The timer is started after sending the message. If the timer expires and we don't receive an answer the message is resent. After one message is lost three times in a row the negotiation is aborted. There is also the possibility of message duplication, which we counter by detecting duplicate messages and discarding them in a limited time window.

## 6. Related work

In **Table 2** we give an overview of existing security agility protocols and compare them by using certain key metrics. The first column analyzes which parameters the communicating parties agree upon. Usually cryptographic key and algorithm agreement should be done in the same exchange, because the algorithms usually can't be used without the appropriate keys. Only the JFK (Just Fast Keying) protocol agrees upon just keys because the algorithm agreement is done in the IPsec protocol. In the communication layer column, we can see that JFK is designed to be used with IPsec. All other protocols except ACAP are bound to a specific layer and they cannot be applied to other communication layers.

**Table 2.** Overview of cryptographically agile protocols

| Protocol name | Key/Algorithm agreement | Communication layer | Scope of solution | Protocol complexity | Formally verified |
|---|---|---|---|---|---|
| ACAP | both | all layers | key and algorithm agreement | low | yes |
| JFK[25] | key | bound to IPsec | key agreement | low | yes |
| IPsec[10] | both | IP layer | traffic protection, VPN | high | partially |
| SSL[11]/TLS[12] DTLS[32] | both | transport layer (stream and datagram) | secure channels | high | partially |
| MinimaLT[14] | both | UDP | secure channels | medium | partially |
| SSH[13] | both | application layer | secure channel and shell | high | partially |
| Tcpcrypt[33] | both | TCP | lightweight secure channels | medium | no |
| QUIC[34] | both | UDP | lightweight secure HTTP transport | medium | no |

The fourth column shows what the protocol is designed to provide. This data is directly connected to the next column. As the scope of the solution grows, so does its complexity. Complexity refers to the number of components that are needed to achieve the scope of the solution.[5] This is a key aspect for security protocols because it makes protocol implementation harder and can be the cause of many pitfalls during protocol lifetime. It also affects the prospects of successful formal modeling and verification. Formal verification serves as an external automated auditing tool and shows that the design ideas in the protocol are satisfied. Formal verification reduces the possibility of future problems in the protocol.

[5] SSL/TLS is composed of four components (Handshake, Change CipherSpec, Alert and Record protocols). IPsec is composed of at least three components (IKE, ESP, SA management plane). SSH is composed of three components (Transport, User Authentication and Connection layer).
MinimalLT, Tcpcrypt and QUIC are lightweight secure channel solutions that have two main components: a key exchange and a secure channel component.
ACAP and JFK are just agreement protocols and consist of only one component.

## 6.1 Short overview of mentioned protocols

**JFK** (Just Fast Keying) [25] proposes an alternative to IKE [35] key exchange protocol that ensures identity protection for either the initiator or the responder. The main differences with ACAP are as follows. We allow for negotiation of cryptographic algorithms. Our communicating parties are equal peers and can be used for all layers and applications. Since identity protection is not included our messages are somewhat simpler and take a different approach while using the same sign-and-mac principles. For our purposes we need two round trip times to establish cryptographic keys and algorithms.

**IPsec** [10] is a cryptographic suite that is used to establish connection between two communicating endpoints through an insecure network. The most widely used scenario is to connect two trusted networks through the Internet. It is comprised of multiple protocols (IKE, ISAKMP, AH, ESP) and cryptographic algorithms. IKE and ISAKMP are implemented as user-space applications but they are tightly connected to the operating system kernel to manage Security Associations and enable proper packet processing. IPsec is used to protect all communication data, but this amount of protection is not always needed and can cause high system load for higher traffic volumes. Updating and maintaining IPsec suite can be time consuming and complex because of the combination of user-space and kernel-space code. **IKE** (Internet Key Exchange) [35] is a standalone protocol that is part of the IPsec protocol suite. It is used for exchanging keys and agreeing upon Security Associations for securing network communication. It is a well known and tested protocol but it is not lightweight and provides a various set of mechanisms. Those mechanisms can be used separately from key exchange and cryptographic algorithm agreement [25].

**SSL/TLS** [11] [12] provides a secure channel over a reliable TCP connection. It is used to protect any data that can be transferred over TCP. Algorithm implementations are integrated with the logic needed to establish a secure channel. SSL and TLS were both designed at the end of the last century and their principles have been established in a period where a lot less was known about secure protocol design and application of cryptographic algorithms. SSL/TLS is a complex system comprised of multiple protocols. Recent attacks prove the vulnerability of the CBC encryption mode [36], use of compression [37], TLS record protocol [38], use of the RC4 stream cipher [39] and introduction of the heartbeat option that enabled the Heartbleed attack [40]. Among design problems TLS implementations recently had problems with certificate management in their implementations e.g. OpenSSL and Apple [41]. If algorithm implementations would have been separated from the protocol logic, library updates would be simpler and less error prone.

**Tcpcrypt** [33] is a novel and lightweight approach to the multiple problems that arose by the usage of SSL/TLS. Tcpcrypt implements a secure channel directly over an established TCP connection. The security handshake process is integrated into the TCP three-way handshake with the addition of only one new message. This greatly reduces the time needed to establish a secure connection. Only 2 round trip times (RTTs) are needed to establish a secure channel, while for a regular TLS session 3 RTTs are needed if the Client Hello message is sent in the last message of the three-way handshake. Server load is also decreased by shifting the RSA encryption to the client. This proves that secure channels can be built in a more efficient manner than SSL/TLS, at least if TCP is used as the transport protocol. In a similar way ACAP solves the distribution of cryptographic prerequisites independently of the application and purpose.

**DTLS** [32] is the adaptation of TLS to datagram communication such as UDP and DCCP. It adds multiple mechanisms to adapt TLS to the datagram paradigm. DTLS allows re-transmissions during the handshake process and arrival of out-of-order data. A similar re-transmission timer is also used in ACAP. DTLS doesn't apply session termination to make secure channels feasible for unreliable transport.

**QUIC** [34] is a new approach for transferring HTTP data over UDP in a secure way but with smaller latencies and system loads. It is integrated into the Chromium browser for faster prototyping and testing. It introduces new mechanisms to prevent IP address spoofing and replay attacks and reduces the number of RTTs amount needed to establish data flow. QUIC adapts the same method of lowering server load by using the same DH exponential for a period of time and pre-computing certain message parts. A similar approach is taken in JFK and ACAP.

**MinimaLT** [14] is a new network protocol that follows a design logic similar to QUIC but is designed to provide higher security and implements novel mechanisms for distributing public keys and DH exchange exponentials by using DNS queries. Communication parties are mutually authenticated by using public keys, which is also applied in ACAP. A new non-interactive key exchange approach [42] is used to minimize RTTs while establishing secure connections. MinimaLT gains more flexibility by transferring all data by UDP instead of TCP but is strictly tied to the transportation layer.

**SSH** [13] presents a widely deployed and generally secure protocol that is used for issuing remote commands, transferring files and tunneling network traffic. It is divided into multiple protocols: transport layer protocol that provides server authentication, confidentiality and integrity, user authentication protocol that authenticates the client and the connection protocol that multiplexes the established secure channel for multiple purposes (command issuing, file transfer, tunneling). The algorithm agreement method used in ACAP is similar to the SSH transport layer protocol. It handles secure distribution of keys and negotiation of cryptographic algorithms.

## 7. Future work

As future work, we plan to evaluate our model and protocol implementation against more attack methods and different attacker models. Our research will also focus on the impact of certain computationally expensive cryptographic operations on the message exchange and their adaptations so that the protocol can be used on devices of varying computing capabilities. In this regard, we are aware of state of the art authenticated key exchange models like HMQV [43] and OAKE [44] and plan to transition from the current SIGMA based exchange to a more lightweight model. We note that this transition must be carried out carefully and formally verified by using a different approach than presented in this paper.

Furthermore, we plan to evaluate the integration of our protocol in widely used and tested protocols. This way, a comprehensive evaluation can be made about the impact of using ACAP on communication latencies. Specifically, we plan to evaluate and compare our implementation with IPsec and SSH mechanisms. Since ACAP is easily adaptable we will also deploy and evaluate it in an environment based on the Internet of Things concept. In that area we want to explore less computationally expensive algorithms that would enable secure communication on devices with less computing power.

## 8. Conclusion

Modern protocols are often burdened with multiple layers and additions, which raise the possibility of vulnerabilities and attacks. In this paper we propose ACAP (Agile Cryptographic Agreement Protocol), that extracts the key and algorithm agreement layer to a dedicated lightweight and adaptable solution. By using formal verification methods we prove that the protocol is secure and safe from the design standpoint. The proposed protocol can be applied to all communication layers in the IP network stack without lowering the security level for communicating parties. Each communicating party is provided with all prerequisites needed for secure communication. This greatly simplifies usage and integration of cryptographic algorithms in previously unsecured protocols and eliminates the need to implement key distribution and management mechanisms. ACAP presents a reliable and integrable building block that can efficiently interoperate with other security mechanisms to provide secure network communication across different network layers, architectures, media and devices.

## References

[1] A. K. Yau, K. G. Paterson, and C. J. Mitchell, “Padding oracle attacks on CBC-mode encryption with secret and random IVs,” *Fast Software Encryption*, pp. 299–319, Springer, Feb. 2005. Article (CrossRef Link)

[2] T. Jager, K. G. Paterson, and J. Somorovsky, “One bad apple: Backwards compatibility attacks on state-of-the-art cryptography,” *Network & Distributed System Security Symposium*, Feb. 2013.

[3] M. Lamberger and F. Mendel, “Higher-Order Differential Attack on Reduced SHA-256,” *IACR Cryptology ePrint Archive*, vol. 2011, p. 37, 2011.

[4] X. Wang, Y. Yin, and H. Yu, “Finding Collisions in the Full SHA-1,” *Advances in Cryptology–CRYPTO 2005*, pp. 17-36, Aug. 2005.

[5] M. Stevens, A. Lenstra, and B. Weger, “Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities,” *Advances in Cryptology-EUROCRYPT 2007*, pp. 1-22, May 2007. Article (CrossRef Link)

[6] P. Sepehrdad, S. Vaudenay, and M. Vuagnoux, “Discovery and Exploitation of New Biases in RC4,” *Selected Areas in Cryptography*, pp. 74–91, Springer, Aug 2010. Article (CrossRef Link)

[7] K. Hu, T. Wolf, T. Teixeira and R. Tessier, "System-level security for network processors with hardware monitors,” *Design Automation Conference (DAC)*, *2014 51st ACM/EDAC/IEEE*, Jun. 2014.

[8] B. Sullivan, “Cryptographic Agility: Defending Against the Sneakers Scenario,” *MSDN Magazine*, Aug. 2009.

[9] M. Howard and S. Lipner, *The security development lifecycle*, O’Reilly Media, Incorporated, 2009.

[10] S. Kent and K. Seo, “Security Architecture for the Internet Protocol,” *RFC 4301 (Proposed Standard)*, Dec. 2005.

[11] A. Freier, P. Karlton, and P. Kocher, “The Secure Sockets Layer (SSL) Protocol Version 3.0,” *RFC 6101 (Historic)*, Aug. 2011.

[12] T. Dierks and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” *RFC 5246 (Proposed Standard)*, Aug. 2008.

[13] T. Ylonen and C. Lonvick, “The Secure Shell (SSH) Protocol Architecture,” *RFC 4251 (Proposed Standard)*, Jan. 2006.

[14] W. M. Petullo, X. Zhang, J. A. Solworth, D. J. Bernstein, and T. Lange, “MinimaLT: minimal-latency networking through better security,” in *Proc. of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 425–438, ACM, 2013. Article (CrossRef Link)
[15] I. Lasc, R. Dojen, and T. Coffey, “On the detection of desynchronisation attacks against security protocols that use dynamic shared secrets,” *Computers & Security*, vol. 32, pp. 115–129, Nov. 2012. Article (CrossRef Link)
[16] A. D. Jurcut, T. Coffey, and R. Dojen, “Design guidelines for security protocols to prevent replay and parallel session attacks,” *Computers and Security*, vol. 45, pp. 255–273, Jun. 2014. Article (CrossRef Link)
[17] V. Vasic, A. Kukec, and M. Mikuc, “Deploying new hash algorithms in secure neighbor discovery,” in *Proc. of 2011 19th International Conference on Software Telecommunications and Computer Networks (SoftCOM)*, Sept. 2011.
[18] J. Arkko, J. Kempf, B. Zill, and P. Nikander, “SEcure Neighbor Discovery (SEND),” *RFC 3971 (Proposed Standard)*, Mar. 2005.
[19] V. Vasic and M. Mikuc, “Security agility solution independent of the underlaying protocol architecture,” in *Proc. of the First International Conference on Agreement Technologies*, Oct. 2012.
[20] H. Krawczyk, “Perfect forward secrecy,” *Encyclopedia of Cryptography and Security*, pp. 921–922, Springer Science & Business Media, 2012.
[21] H. Krawczyk, “Sigma: The ‘SIGn-and-MAc’ Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols,” *Advances in Cryptology-CRYPTO 2003*, pp. 400–425, Springer, Aug. 2003. Article (CrossRef Link)
[22] W. Diffie and M. Hellman, “New directions in cryptography,” *Information Theory, IEEE Transactions on*, vol. 22, pp. 644–654, Nov 1976. Article (CrossRef Link)
[23] M. Bellare, R. Canetti, and H. Krawczyk, “Keying hash functions for message authentication,” *Advances in Cryptology — CRYPTO ’96*, pp. 1–15, Springer Berlin Heidelberg, 1996. Article (CrossRef Link)
[24] H. Krawczyk, M. Bellare, and R. Canetti, “HMAC: Keyed-Hashing for Message Authentication,” *RFC 2104 (Informational)*, Feb. 1997.
[25] W. Aiello, S. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. Keromytis, and O. Reingold, “Just fast keying: Key agreement in a hostile internet,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 2, pp. 242–273, May 2004. Article (CrossRef Link)
[26] T. Ylonen and C. Lonvick, “The Secure Shell (SSH) Transport Layer Protocol.,” *RFC 4253 (Proposed Standard)*, Jan. 2006.
[27] R. Canetti and H. Krawczyk, “Security Analysis of IKE’s Signature-Based Key-Exchange Protocol,” *Advances in Cryptology—CRYPTO 2002*, pp. 143–161, Springer, Aug. 2002. Article (CrossRef Link)
[28] C. J. F. Cremers, *Scyther: Unbounded Verification of Security Protocols*, ETH, Department of Computer Science, 2007.
[29] C. J. F. Cremers, *Scyther user manual*, 2014.
[30] C. J. F. Cremers, *Scyther: Semantics and verification of security protocols*, Eindhoven University of Technology, 2006.
[31] G. Lowe, “A hierarchy of authentication specifications,” in *Proc. of 10th Computer Security Foundations Workshop*, pp. 31–43, IEEE, 1997. Article (CrossRef Link)
[32] E. Rescorla and N. Modadugu, “Datagram Transport Layer Security Version 1.2.,” *RFC 6347 (Proposed Standard)*, Jan. 2012.
[33] A. Bittau, M. Hamburg, M. Handley, D. Mazieres, and D. Boneh, “The case for ubiquitous transport-level encryption.,” *USENIX Security Symposium*, pp. 403–418, Aug. 2010.
[34] J. Roskind, QUIC (Quick UDP Internet Connections): Multiplexed Stream Transport Over UDP, Dec. 2013.
[35] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, “Internet Key Exchange Protocol Version 2 (IKEv2).,” *RFC 7296 (INTERNET STANDARD)*, Oct. 2014.

[36] T. Duong and J. Rizzo, “Here come the XOR Ninjas,” *White paper, Netifera*, 2011.
[37] T. Be’ery and A. Shulman, “A perfect crime? only time will tell,” *Black Hat Europe 2013*, Mar. 2013.
[38] N. J. A. Fardan and K. Paterson, “Lucky Thirteen: Breaking the TLS and DTLS Record Protocols,” *Security and Privacy (SP), 2013 IEEE Symposium on*, pp. 526–540, IEEE, May 2013. Article (CrossRef Link)
[39] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. Schuldt, “On the Security of RC4 in TLS,” *USENIX Security Symposium*, pp. 305–320, Aug. 2013.
[40] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer et al., “The Matter of Heartbleed,” in *Proc. of the 2014 Conference on Internet Measurement Conference*, pp. 475–488, ACM, May 2014. Article (CrossRef Link)
[41] M. Bland, “Finding More Than One Worm in the Apple,” *ACM Queue vol. 12*, pp. 10:10–10:21, May 2014.
[42] E. S. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson, “Non-Interactive Key Exchange,” *Public-Key Cryptography–PKC 2013*, pp. 254–271, Springer, 2013. Article (CrossRef Link)
[43] H. Krawczyk, “HMQV: A High-Performance Secure Diffie-Hellman Protocol,” in *Proc. of the 25th Annual International Conference on Advances in Cryptology, CRYPTO’05*, pp. 546–566, Springer-Verlag, 2005. Article (CrossRef Link)
[44] A. C.-C. Yao and Y. Zhao, “OAKE: a new family of implicitly authenticated diffie-hellman protocols,” in *Proc. of the 2013 ACM SIGSAC conference on Computer & communications security, CCS ’13*, pp. 1113–1128, ACM, Apr. 2013. Article (CrossRef Link)

**Valter Vasić** received his M.Sc. degree in information and communication technology from University of Zagreb in 2010. He is a PhD student and research associate at the University of Zagreb, Faculty of Electrical Engineering and Computing. He was researcher on the E-IMUNES project funded by Ericsson Nikola Tesla, Zagreb. He published more than 10 papers in journals and conference proceedings. The focus of his research is secure network communication. His research interests are in the area of network communication, computer security and virtualization.

**Miljenko Mikuc** M.Sc. and Ph.D. degrees in electrical engineering from the University of Zagreb, Faculty of Electrical Engineering and Computing. From 1988 he is working at the Department of Telecommunications at FER. In December 2004 he was promoted to Associate Professor. He was a project leader of 2 projects of Applications of Information Technology financed by the Ministry of Science, Education and Sports of the Republic of Croatia and the project leader of cooperation research projects with "The Boeing Company", "International Computer Science Institute" and "The FreeBSD Foundation" from USA and with Ericsson Nikola Tesla d.d. from Zagreb. Currently he is a project leader of the research project: „Ericsson Customized IMUNES (E-IMUNES)“ in cooperation with Ericsson Nikola Tesla d.d. company. He published more than 30 papers in journals and conference proceedings in the area of communication networks, protocols, virtualization, formal methods and security.

**Marin Vuković** is an Assistant Professor at the Telecommunication Department of the Faculty of Electrical Engineering and Computing, University of Zagreb. He graduated at the Faculty of Electrical Engineering and Computing, University of Zagreb in 2006 and received a PhD in 2011. In scientific and professional work Marin Vukovic focuses on value added services as well as advanced services in converged next generation networks based on knowledge, and participates as a researcher and associate in scientific, technological and professional projects. He published over 20 scientific papers. He is a co-author of the patent at the Croatian Institute for Intellectual Property P20080303A.