Zero Round-Trip Time for the Extended Access Control Protocol

Jacqueline Brendel(✉) and Marc Fischlin

Cryptoplexity, Technische Universität Darmstadt, Darmstadt, Germany
{jacqueline.brendel,marc.fischlin}@cryptoplexity.de
http://www.cryptoplexity.de

Abstract. The Extended Access Control (EAC) protocol allows to create a shared cryptographic key between a client and a server. While originally used in the context of identity card systems and machine readable travel documents, the EAC protocol is increasingly adopted as a universal solution to secure transactions or for attribute-based access control with smart cards. Here we discuss how to enhance the EAC protocol by a so-called zero-round trip time (0RTT) mode. Through this mode the client can, without further interaction, immediately derive a new key from cryptographic material exchanged in previous executions. This makes the 0RTT mode attractive from an efficiency viewpoint such that the upcoming TLS 1.3 standard, for instance, will include its own 0RTT mode. Here we show that also the EAC protocol can be augmented to support a 0RTT mode. Our proposed EAC+0RTT protocol is compliant with the basic EAC protocol and adds the 0RTT mode smoothly on top. We also prove the security of our proposal according to the common security model of Bellare and Rogaway in the multi-stage setting.

1 Introduction

The *Extended Access Control* (EAC) protocol establishes an authenticated key between a client's smart card (also called chip in this context) and a server (or, terminal) over a public channel. For this, both parties run a sophisticated Diffie-Hellman key exchange protocol in which either party deploys its certified long-term key. While originally deployed in the German identity card systems [10] and referenced by the International Civil Aviation Organization for machine readable travel documents [24], the EAC protocol is increasingly adopted as a potent solution in related scenarios, for example to secure transactions [29] and for attribute-based physical access control with smart cards [28].

Especially for access control, if deployed in situations where user experience hinges on fast response times, reducing the latency is important. A concrete example, as discussed in a FIPS 201-2 workshop in 2015 [19], is turnstile access in subway stations. This requirement has led for instance to the development of the ISO/IEC 24727-6 and ANSI 504-1 standardized "Open Protocol for Access Control Identification and Ticketing with privacY" (OPACITY) for smart cards [33], which uses persistent binding for speeding up the key generation process.

S.N. Foley et al. (Eds.): ESORICS 2017, Part I, LNCS 10492, pp. 297–314, 2017.
DOI: 10.1007/978-3-319-66402-6_18

Unfortunately—and also underlining the importance of rigor—OPACITY has been shown to display cryptographic weaknesses [15]. [1]

In this paper we show that the EAC protocol can be augmented by a low-latency mode, called *zero round-trip time* (0RTT). This mode enables efficient re-establishment of secure channels for returning clients. A rigorous security proof for the resulting augmented protocol completes the enhancement. We emphasize that the design choices of the original EAC protocol are beyond our discussion here. Our goal is to show that a 0RTT version can be implemented based on the existing infrastructure.

1.1 Striving for Zero Round-Trip Time

The EAC protocol consists of two connected phases, the terminal authentication (TA), followed by the chip authentication (CA). Both steps require only a small number of message exchanges to establish a session key. At the same time, recent efforts in the area of key exchange protocols aim at modes of operations which allow for even faster data delivery. More precisely, it should be possible for a party to re-use cryptographic data from a previous connection to derive a fresh session key without further interaction, thus allowing the party to transmit data immediately. Such a mode is called *zero round-trip time* (0RTT).

The first proposal for a 0RTT-supporting protocol came from Google with its QUIC protocol [20]. The 0RTT mode allows the client to send data to a known server without having to wait for the server's response. This idea was then quickly adopted for the drafts of the new TLS version 1.3, and has been included in the latest drafts in various versions [30–32]. Even on a network layer level, the Windows Networking Team recently announced to support 0RTT for TCP connections in order to reduce latency (see [11] for TCP Fast Open description).

The rough idea of the approach taken by QUIC and TLS (for the Diffie-Hellman version [30]) [2] is that, upon the first encounter, the server also sends a semi-static public key g^s as part of the authenticated key exchange. Unlike an ephemeral key, which is used only within a single session, and a long-term key which spans over a large amount of sessions, such a semi-static key is valid for a very limited time only. This time period may range from a few seconds to a couple of days. In particular, the semi-static key may be used in multiple sessions.

The next time the client contacts the server, the client may combine a fresh ephemeral key g^c with the server's semi-static key g^s to immediately compute a Diffie-Hellman key g^{cs} and derive an intermediate session key. The client can now send g^c and already deliver data secured under the intermediate session key,

[1] Remarkably, the publication of this analysis pre-dates the latest version of SP800-73-4 [12], dated May 2015, which lists OPACITY as a suitable solution for key establishment.

[2] The latest version of the TLS draft [32] focuses on a pre-shared key 0RTT version and has for now dropped the Diffie-Hellman based version; the main EAC protocol only supports a Diffie-Hellman based key exchange, though.

without round trip. For both QUIC and TLS the parties then continue the key exchange protocol to switch to full session keys.

It is obvious that the non-interactive derivation of the 0RTT session key comes at a price in terms of security: Since the server cannot contribute to such a key in a per-session manner, an adversary can replay the client's protocol message and data to the server. This is inevitable, but accepted by the designers of QUIC and TLS 1.3 as worthwhile to achieve the desired level of efficiency.

1.2 Contribution

As briefly mentioned before, we show that the EAC protocol can also be augmented to support a 0RTT mode. Interestingly, the extension can be added on top with minimal changes to the original protocol. As in the proposal of QUIC and TLS 1.3 we let the terminal include an additional semi-static key pk_T^{semi} in the regular EAC execution. The key is transmitted as part of the auxiliary data field of the original EAC description, and is thus also authenticated through the terminal's signature in the TA phase.

In the full run of the EAC protocol the semi-static key is still ignored for the session key derivation. Instead, and as in the original EAC description, the chip then receives the terminal's ephemeral key and derives a session key from its certified long-term key and this ephemeral key. The client authenticates through a message authentication code under the session key. In this regard, the slightly modified protocol complies with the original EAC protocol, using the auxiliary data field to transfer an additional key.

If a chip later wants to reconnect to a terminal for which it already holds the semi-static key, it only runs the CA phase again. But instead of receiving a fresh ephemeral key from the terminal, it uses the semi-static key to build the session key. Note that the semi-static key is already authenticated through the previous execution of the EAC protocol. Omitting the transmission of the terminal's ephemeral key turns this step into a non-interactive protocol.

A straightforward idea to improve efficiency further may be to use the terminal's ephemeral key once more for 0RTT, instead of using the semi-static key. The downside is that the terminal would need to store all ephemeral keys in a certain time frame. This is why, both we here as well as TLS [30], use semi-static keys instead. Nonetheless we discuss some potential variations of our basic designs in Sect. 4.

We then show that our EAC+0RTT protocol, which consists of the (augmented) EAC protocol run followed by any number of subsequent 0RTT EAC protocol executions, meets the common security properties of an authenticated key exchange protocol.

But we, of course, need to account for the possibility of replay attacks on the 0RTT data. Furthermore, it is convenient to model the possibly many 0RTT EAC handshakes following a single EAC execution in a so-called *multi-stage* setting. To this end we adopt the multi-stage extension of the Bellare-Rogaway model in [17].

The proof of security for the EAC+0RTT protocol does not rely on previous results. Nevertheless, we wish to mention the many security analyses of the German identity card protocols and certain eIDAS extensions [2–5,13,14,22,23,27]. Also, we remark that general approaches to build low-latency protocols such as [21] cannot be applied in the context of the EAC protocol without major changes to the protocol.

2 Protocol Description

We next present the Extended Access Control protocol and its extension to support 0RTT. The 0RTT extension should be seen as a particular mode or sub protocol which co-exists with the original EAC protocol. In particular, many instances of 0RTT EAC may follow a single full EAC protocol run (until pk_T^{semi} changes, in which case the terminal will most likely reject).

2.1 The Extended Access Protocol

The Extended Access Control protocol establishes a secure channel between a chip and a terminal. It is divided in two phases: the Terminal Authentication (TA) and Chip Authentication (CA) as depicted in Fig. 1. We integrate the 0RTT EAC protocol to the existing EAC protocol smoothly by using the pre-specified auxiliary data field in which any data can be sent in an authenticated manner to the chip during the TA phase. The auxiliary data field has originally been included to pass further information to the chip such as the current date, and the original EAC protocol ignores any such data if sent under an unknown object identifier. In our case, the terminal can utilize this field to transmit its semi-static key pk_T^{semi} to the chip to enable future 0RTT EAC executions.

Terminal Authentication. The terminal authentication lets the chip C verify the terminal T's identity and its permissions to access sensitive data. This is achieved via the certificate $cert_T$ held by T. This certificate contains not only the terminal's signed public key but also its granted access rights. We assume that each certificate *cert* contains some unique identifier *certID* which can either be the serial number or an identifier like `CertID` or `CertUID`, and that *certID* allows to determine the user identity. Furthermore, as mentioned earlier, the terminal authentication can be used to distribute the terminal's public semi-static key to the chip, thereby permitting future 0RTT EAC executions.

In a first step, the terminal sends its certificate for verification to the chip, which can then either abort, in case of an invalid certificate, or proceed by extracting the terminal's public key pk_T from the valid certificate. If the session was not aborted by C, T generates its ephemeral key pair (epk_T, esk_T) and sends the compressed version of the ephemeral key epk_T to C. This initiates a challenge-response mechanism. The chip replies with a nonce r_C chosen uniformly at random. The terminal authentication is complete, if the chip can then successfully verify the received signature $s_T \leftarrow \mathsf{Sig}(sk_T, id_C||r_C||\mathsf{Compr}(epk_T)||pk_T^{\mathsf{semi}})$ over

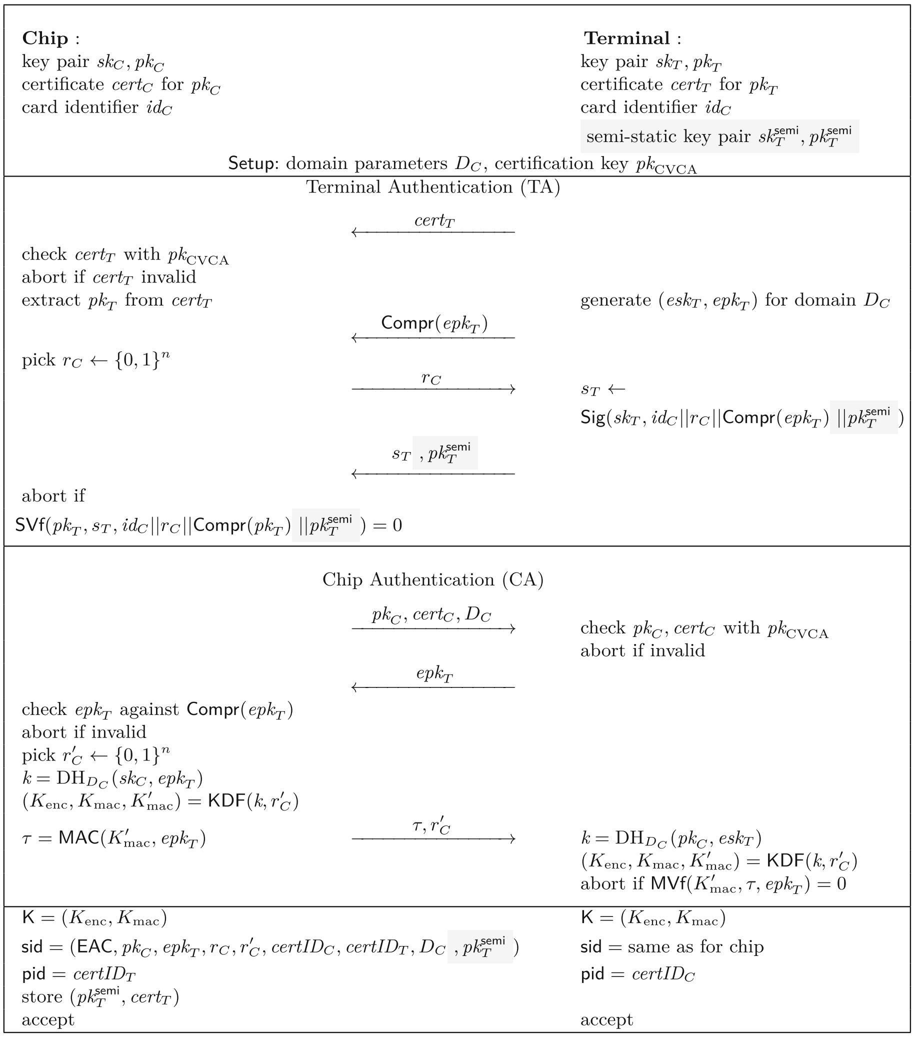
Chip :
key pair sk_C, pk_C
certificate $cert_C$ for pk_C
card identifier id_C
Terminal :
key pair sk_T, pk_T
certificate $cert_T$ for pk_T
card identifier id_C
semi-static key pair $sk_T^{\mathsf{semi}}, pk_T^{\mathsf{semi}}$
Setup: domain parameters D_C, certification key pk_{CVCA}
Terminal Authentication (TA)
$\xleftarrow{cert_T}$
check $cert_T$ with pk_{CVCA}
abort if $cert_T$ invalid
extract pk_T from $cert_T$
generate (esk_T, epk_T) for domain D_C
$\xleftarrow{\mathsf{Compr}(epk_T)}$
pick $r_C \leftarrow \{0,1\}^n$
$\xrightarrow{r_C}$
$s_T \leftarrow$
$\mathsf{Sig}(sk_T, id_C||r_C||\mathsf{Compr}(epk_T)\ ||pk_T^{\mathsf{semi}}\ )$
$\xleftarrow{s_T\ , pk_T^{\mathsf{semi}}}$
abort if
$\mathsf{SVf}(pk_T, s_T, id_C||r_C||\mathsf{Compr}(pk_T)\ ||pk_T^{\mathsf{semi}}\ ) = 0$
Chip Authentication (CA)
$\xrightarrow{pk_C, cert_C, D_C}$
check $pk_C, cert_C$ with pk_{CVCA}
abort if invalid
$\xleftarrow{epk_T}$
check epk_T against $\mathsf{Compr}(epk_T)$
abort if invalid
pick $r'_C \leftarrow \{0,1\}^n$
$k = \text{DH}_{D_C}(sk_C, epk_T)$
$(K_{\text{enc}}, K_{\text{mac}}, K'_{\text{mac}}) = \mathsf{KDF}(k, r'_C)$
$\tau = \mathsf{MAC}(K'_{\text{mac}}, epk_T)$
$\xrightarrow{\tau, r'_C}$
$k = \text{DH}_{D_C}(pk_C, esk_T)$
$(K_{\text{enc}}, K_{\text{mac}}, K'_{\text{mac}}) = \mathsf{KDF}(k, r'_C)$
abort if $\mathsf{MVf}(K'_{\text{mac}}, \tau, epk_T) = 0$
$\mathsf{K} = (K_{\text{enc}}, K_{\text{mac}})$
$\mathsf{sid} = (\mathsf{EAC}, pk_C, epk_T, r_C, r'_C, certID_C, certID_T, D_C\ , pk_T^{\mathsf{semi}}\ )$
$\mathsf{pid} = certID_T$
store $(pk_T^{\mathsf{semi}}, cert_T)$
accept
$\mathsf{K} = (K_{\text{enc}}, K_{\text{mac}})$
sid = same as for chip
$\mathsf{pid} = certID_C$
accept

Fig. 1. Terminal Authentication (TA) and Chip Authentication (CA). All operations are modulo q resp. over the elliptic curve. The gray part shows the 0RTT support inserted in the (optional) auxiliary data field.

the chip's identity, chosen nonce and the compressed ephemeral key. Depending on whether the terminal offers support for 0RTT executions, the signature may contain the terminal's semi-static public key pk_T^{semi}.

Chip Authentication. In the second part of the EAC protocol, the chip is authenticated to the terminal and a session key for subsequent encrypted and integrity-protected communications between chip and terminal is established.

The chip transmits its credentials to the terminal and receives in response the ephemeral public key epk_T (if the terminal did not abort due to an invalid certificate). After checking epk_T against the compressed value received during the TA phase, the chip can compute the Diffie-Hellman value k from epk_T and its own long-term secret key sk_C. Together with a uniformly random value r'_C, the DH value k is used to derive an encryption key K_{enc}, as well as two authentication keys $K_{\mathrm{mac}}, K'_{\mathrm{mac}}$. [3] For final authentication, the chip uses K'_{mac} to compute a tag τ over the ephemeral public key of the terminal. The tag is then transmitted to the terminal, alongside the random value r'_C used in the key derivation. The terminal is now able to derive the DH key k and subsequently the keys $(K_{\mathrm{enc}}, K_{\mathrm{mac}}, K'_{\mathrm{mac}})$, where the session key K is given by $(K_{\mathrm{enc}}, K_{\mathrm{mac}})$. The terminal aborts the CA phase prematurely if it is not able to verify τ. Otherwise the session identifier and partner identifier are generated on both sides. If C has received a semi-static key, it saves this key along with the terminal's certificate $cert_T$ for further reference. The EAC protocol execution is completed successfully if both parties terminate in accepting state.

2.2 The 0RTT EAC Protocol

Figure 2 shows the modified protocol supporting 0RTT between a chip C and a terminal T. The chip now holds additional information in form of the semi-static public key pk_T^{semi}, which it obtained during a previous EAC protocol interaction with T. In the 0RTT extension of the EAC protocol, C and T perform the following actions, corresponding to a non-interactive version of the CA protocol since the pk_T^{semi} is used instead of epk_T. Thus, the extra communication round in the CA protocol in which T sends the (uncompressed) ephemeral key becomes obsolete.

At first, the chip C picks a random nonce r''_C and computes the DH shared value $k = \mathrm{DH}_{D_C}(sk_C, pk_T^{\mathsf{semi}})$. Using these two values, C then derives the keys $(K_{\mathrm{enc}}, K_{\mathrm{mac}}, K'_{\mathrm{mac}})$ where, as in the EAC protocol, K'_{mac} is an additional authentication key used internally in the 0RTT EAC key exchange (see [14] for a discussion). The session key is then given by $\mathsf{K} = (K_{\mathrm{enc}}, K_{\mathrm{mac}})$. Finally, C computes the MAC-value over the semi-static public key

$$\tau = \mathsf{MAC}(K'_{\mathrm{mac}}, pk_T^{\mathsf{semi}})$$

and sends its first (and only) flight of data to T consisting of

- the authentication token τ,
- the previously chosen nonce r''_C,
- its public key pk_C, as well as its certificate $cert_C$,
- the domain parameter D_C, and
- early application data encrypted under the previously derived key.

[3] For the necessity of K'_{mac} in a proof in the Bellare-Rogaway-style we refer to the discussion in [14].

Upon receiving the chip's message, T verifies the validity of pk_C and $cert_C$, and aborts if the verification is unsuccessful. Otherwise, T uses the public key, its semi-static secret sk_T^{semi} and the random nonce r''_C to derive K'_{mac} and the 0RTT EAC session key K. T can then check the validity of the authentication token τ and aborts if the tag cannot be verified. If τ is valid, T decrypts the attached early application data. This completes the 0RTT EAC execution.

If the terminal does not support 0RTT, or the semi-static key provided by the chip is outdated or otherwise invalid, the process is aborted and the chip must initiate a fresh execution of the full EAC protocol in order to establish an authenticated secure channel with the terminal. There are, of course, several conceivable ways to recover from failures in the 0RTT handshake. Possible alternatives are described in Sect. 4.3.

Chip :		**Terminal** :
key pair sk_C, pk_C		key pair sk_T, pk_T
certificate $cert_C$ for pk_C		certificate $cert_T$ for pk_T
card identifier id_C		card identifier id_C
semi-static public key $(pk_T^{\mathsf{semi}}, cert_T)$		semi-static key pair $sk_T^{\mathsf{semi}}, pk_T^{\mathsf{semi}}$
	Setup: domain parameters D_C, certification key pk_{CVCA}	
	Zero Round-Trip Time (0RTT)	
pick $r''_C \leftarrow \{0,1\}^n$		
$k = \mathrm{DH}_{D_C}(sk_C, pk_T^{\mathsf{semi}})$		
$(K_{\mathrm{enc}}, K_{\mathrm{mac}}, K'_{\mathrm{mac}}) = \mathsf{KDF}(k, r''_C)$		
$\tau = \mathsf{MAC}(K'_{\mathrm{mac}}, pk_T^{\mathsf{semi}})$	$\xrightarrow{\tau, r''_C, pk_C, cert_C, D_C}$	check $pk_C, cert_C$ with pk_{CVCA}
		abort if invalid
		$k = \mathrm{DH}_{D_C}(pk_C, s)$
		$(K_{\mathrm{enc}}, K_{\mathrm{mac}}, K'_{\mathrm{mac}}) = \mathsf{KDF}(k, r''_C)$
		abort if $\mathsf{MVf}(K'_{\mathrm{mac}}, \tau, pk_T^{\mathsf{semi}}) = 0$
$\mathsf{K} = (K_{\mathrm{enc}}, K_{\mathrm{mac}})$		$\mathsf{K} = (K_{\mathrm{enc}}, K_{\mathrm{mac}})$
$\mathsf{sid} = (\mathsf{0RTT}, r''_C, pk_C, pk_T^{\mathsf{semi}}, certID_C, certID_T, D_C)$		sid = same as for chip
$\mathsf{pid} = certID_T$		$\mathsf{pid} = certID_C$
accept		accept

Fig. 2. 0RTT EAC. All operations are modulo q resp. over the elliptic curve. Note that the fields sid and pid are used within the security proof and describe partnered sessions and intended communication partners.

2.3 Discussion

As mentioned before, the design choices of the original EAC protocol are beyond our discussion here. We demonstrated that a 0RTT version can be implemented based on the existing infrastructure. In particular, it is important that such a solution is "non-invasive" in the sense that it does not require major changes to the existing protocol but is added "on top". Of course, any extension brings some modifications, e.g., in our case both the chip and the terminal must now implement the 0RTT EAC protocol and store semi-static keys. Yet, our proposal for the augmented EAC protocol complies with the original EAC description by using the auxiliary data field for the semi-static key. Furthermore, the 0RTT mode is identical to the plain execution of the CA phase, only that the semi-static key identifier is used instead of the one for the ephemeral key.

We also stress that we do not comment on the security-efficiency trade-off concerning 0RTT modes, but rather offer the option to have such a mode for the EAC protocol in principle. Whether chips and terminals eventually support this mode and tolerate for example the replay problem, is case dependent. Still, the examples of QUIC and TLS 1.3 indicate that, from an engineering perspective, the desire to have such modes exists, and we provide a potential technical solution for EAC.

Finally, let us point out that 0RTT transfers inherently include the small risk that the transmitted data cannot be recovered by the receiver, e.g., if the receiver has switched the semi-static key in the meantime. For common client-server scenarios the client may thus have to re-transmit the data. This problem is often outweighed by the efficiency gain in the regular cases. For smart card applications it may be preferable to have the terminal first signal its support of 0RTT and to communicate the current identifier of the semi-static key, thus saving the card from performing unnecessary operations. This can be done with the transmission of the certificate in the first step of the TA protocol, allowing the card to decide which mode to execute. Strictly speaking, this would effectively support a "lightweight 1RTT" protocol mode, still with significant efficiency advantages.

3 Overview over Security Analysis

Due to space restrictions we only give a brief overview over our security results. A comprehensive description of the model and the complete security proofs are available in the full version [6].

3.1 Game-Based Approach

The main theorem (Theorem 2) is proven by a technique commonly referred to as *game-hopping*. The proof is organized as a finite sequence of games $G_0, G_1, \ldots, G_k$ which are played between a *challenger* and an *adversary*. Informally, the transitions from one game to the next are small changes to the environment in which the adversary is situated, leading from a position where the winning probability of the attacker is unknown (game G_0) to a situation where this probability can be determined (game G_k). The overall goal is to bound the adversary's advantage in winning the original security game G_0 by the inverse of any polynomial in the security parameter.

3.2 Security Model

The security model is situated within the game-based approach of Bellare and Rogaway (BR model) [1] in which an adversary with full control over the network, must be able to distinguish real session keys from independently drawn keys.

To this end, the adversary can interact with protocol participants and instances via oracles. Details on these queries follow shortly.

A single execution of EAC between a chip and a terminal may be followed by multiple 0RTT handshakes between the parties. To model this situation, we adopt the notion of *multi-stage* key exchange as originally introduced in the related QUIC analysis of Fischlin and Günther [17]. This extension of the BR model allows for multiple keys to be established within a single session. As opposed to the multi-stage setting encountered in e.g. QUIC, we can make use of a simplified setting here, since no key derived within a session is used to secure communications in further stages of the same session. Thus, all keys derived in a single session can be seen as independent.

Adversarial Interaction. To initiate a new session the adversary can call the NewSession oracle, which takes a label to determine which of the two modes (full EAC or 0RTT EAC) to execute. The adversary can query the Send oracle to send protocol message to an instance, immediately getting the party's reply in return. The adversary is furthermore permitted to learn the long-term secret keys of parties through a Corrupt oracle. Leakage of session keys and semi-static secret keys, which are used to derive 0RTT session keys, is modeled through Reveal and RevealSemiStaticKey queries, respectively.

To engage with the BR game (cf. Definition 2), the adversary may perform Test queries for some session(s) of the protocol, resulting in either the receipt of the corresponding session key or of an independently and uniformly chosen key, the choice made at random. In order to win the game, the adversary must now distinguish which kind of key it received.

Freshness of Session Keys. In order to avoid trivial attacks, some restrictions concerning the Test queries apply. Foremost, the party of a tested session must not be corrupt, or else the adversary is trivially able to compute the session key. Analogously, neither the tested session key may have been revealed to the adversary nor the party's semi-static secret key in case of the 0RTT mode. Since both communication parties are supposed to derive the same session key in a key exchange protocol, we must also rule out similar trivial attacks on the communication partner of a tested session. To keep track if one of these cases has occurred, a flag lost is introduced with initial value false. Here, communication partners are usually identified through session identifiers which determine sessions belonging together.

Security Definitions. We follow the approach of Brzuska et al. [8,9], and Fischlin and Günther [17], and separate the required security properties into Match security and BR security. The conditions on Match security guarantee that the session identifiers enable the correct identification of partnered sessions, while partner identifiers pid reflect the correct intended communication partners. Multi-Stage BR security refers to Bellare-Rogaway-like key secrecy as discussed above, demanding that for each stage, session keys appear to be fresh random keys.

The subsequent analysis of the EAC+0RTT protocol is based on the following security notions as described in [16] and adapted to our particular setting:

Definition 1 (Match security). *Let n be the security parameter. Furthermore let* KE *be a key exchange protocol and let* $\mathcal{A}$ *be a PPT adversary interacting with* KE *in the following game* $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}(n)$*:*

Setup. *The challenger generates long-term public/private-key pairs with certificates for each participant* $U \in \mathcal{U}$.

Query. *The adversary* $\mathcal{A}$ *receives the generated public keys and has access to the queries* NewSession, NewSemiStaticKey, Send, Reveal, RevealSemiStaticKey, *and* Corrupt.

Stop. *At some point, the adversary stops with no output.*

We say that $\mathcal{A}$ *wins the game, denoted by* $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}(n) = 1$, *if at least one of the following conditions holds:*

1. *There exist two labels* label, label′ *and stages* $i, j \in \{1, \ldots, M\}$ *such that* $(\mathsf{label}, i) \neq (\mathsf{label}', j)$ *but* $\mathsf{sid}_i = \mathsf{sid}'_j \neq \bot$, $\mathsf{label}.stage \geq i$, $\mathsf{label}'.stage \geq j$ *and* $\mathsf{st}_{\mathsf{exec},i} \neq \mathsf{rejected}$, *and* $\mathsf{st}_{\mathsf{exec},j'} \neq \mathsf{rejected}$, *but* $\mathsf{K}_i \neq \mathsf{K}'_i$. *(Different session keys in partnered sessions, either within the same session at different stages or across two sessions.)*
2. *There exist two labels* label, label′ *such that* $\mathsf{sid}_i = \mathsf{sid}'_j \neq \bot$ *for some stages* $i, j \in \{1, \ldots \mathsf{M}\}$, $\mathsf{role} = \mathsf{initiator}$, *and* $\mathsf{role}' = \mathsf{responder}$, *but* $\mathsf{label.ownid} \neq \mathsf{label}'.\mathsf{pid}$ *or* $\mathsf{label.pid} \neq \mathsf{label}'.\mathsf{ownid}$. *(Different intended partner.)*
3. *There exist at least three labels* label, label′ *and* label″ *and stages* i, j, k *such that* (label, i), (label', j), (label'', k) *are pairwise distinct, but* $\mathsf{sid}_i = \mathsf{sid}'_j = \mathsf{sid}''_k \neq \bot$ *and for any two of the three sessions with role* responder *and mode* 0RTT *it holds that the owners are distinct. (More than two sessions share a session id for some stage and this event was not caused by a simple replay attack on the 0RTT protocol for the same responder.)*

We say KE *is* Match-secure *if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter* n*:* $\mathsf{Adv}^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}(n) = 1\right]$.

Definition 2 (BR Key Secrecy). *Let n be the security parameter. Furthermore let* KE *be a key exchange protocol with key distribution* $\mathcal{D}$ *and let* $\mathcal{A}$ *be a PPT adversary interacting with* KE *in the following game* $G^{\mathsf{BR},\mathcal{D}}_{\mathsf{KE},\mathcal{A}}(n)$*:*

Setup. *The challenger generates long-term public/private-key pairs and certificate for each participant* $U \in \mathcal{U}$, *chooses the test bit* $b_{\mathsf{test}} \xleftarrow{\$} \{0,1\}$ *at random, and sets* $\mathsf{lost} \leftarrow \mathsf{false}$.

Query. *The adversary* $\mathcal{A}$ *receives the generated public keys and has access to the queries* NewSession, NewSemiStaticKey, Send, Reveal, RevealSemiStaticKey, Corrupt, *and* Test.

Guess. *At some point,* $\mathcal{A}$ *stops and outputs a guess* b_{guess}.

Finalize. *The challenger sets the 'lost' flag to* $\mathsf{lost} \leftarrow \mathsf{true}$ *if there exist two (not necessarily distinct) labels* $\mathsf{label}, \mathsf{label}'$ *and stages* $i, j \in \{1, \ldots, \mathsf{M}\}$ *such that* $\mathsf{sid}_i = \mathsf{sid}'_j$, $\mathsf{label.st}_{\mathsf{key},i} = \mathsf{revealed}$, *and* $\mathsf{label}'.\mathsf{tested}j = \mathsf{true}$. *(Adversary has tested and revealed the key in a single session or in two partnered sessions.)*

$\mathcal{A}$ *wins the game, denoted by* $G^{\mathsf{BR},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} = 1$, *if* $b_{\mathsf{guess}} = b_{\mathsf{test}}$ *and* $\mathsf{lost} = \mathsf{false}$. *We say that* Multi-Stage BR key secrecy *holds for* KE *if for all PPT adversaries* $\mathcal{A}$ *the advantage function*

$$\mathsf{Adv}^{\mathsf{BR},\mathcal{D}}_{\mathsf{KE},\mathcal{A}}(n) := \Pr\left[G^{\mathsf{BR},\mathcal{D}}_{\mathsf{KE},\mathcal{A}}(n) = 1\right] - \frac{1}{2}$$

is negligible in the security parameter n. *A key exchange protocol* KE *is further called* Multi-Stage BR-secure *if* KE *is both* Match*-secure and* BR *key secrecy for* KE *holds.*

We note that the winning conditions are independent of the forward secrecy property of the KE protocol. Forward secrecy is already taken into account in the formulation of the Reveal and Corrupt queries and the finalization step of the game.

3.3 Cryptographic Assumptions

In the following we will provide definitions of the basic cryptographic assumptions underlying the security proof of the EAC+0RTT protocol. In particular, we introduce a double-sided (or symmetric) variant of the PRF-ODH assumption, further referred to as mmPRF-ODH. We start by recalling what it means for signatures and certificates to be existentially unforgeable under chosen message attacks:

Definition 3 (EUF-CMA assumption). *Let* n *be the security parameter. Furthermore let* $\mathcal{S} = (\mathit{SKG}, \mathit{Sig}, \mathit{SVf})$ *be a signature scheme and let* $\mathcal{A}$ *be a PPT algorithm. We define the following* EUF-CMA *security game* $G^{\mathsf{EUF\text{-}CMA}}_{\mathit{Sig},\mathcal{A}}(n)$*:*

Setup. *Generate a key pair* $(pk, sk) \xleftarrow{\$} \mathit{SKG}(1^n)$ *and give* pk *to the adversary* $\mathcal{A}$.

Query Phase. *In the next phase* $\mathcal{A}$ *can adaptively query messages* $m_1, m_2, \ldots, m_q \in \{0,1\}^*$ *with* $q \in \mathbb{N}$ *arbitrary, which the signing oracle answers with* $\sigma_1 \leftarrow \mathit{Sig}(sk, m_1), \sigma_2 \leftarrow \mathit{Sig}(sk, m_2), \ldots, \sigma_q \leftarrow \mathit{Sig}(sk, m_q)$.

Output. *At some point,* $\mathcal{A}$ *outputs a message* m^* *and a potential signature* σ^*. *Output* 1 *iff* $\mathit{SVf}(pk, m^*, \sigma^*) = 1$ *and* $m^* \neq m_i$ *for all* $i = 1, 2, \ldots, q$.

We define the advantage function

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{S},\mathcal{A}}(n) := \Pr\left[G^{\mathsf{EUF\text{-}CMA}}_{\mathit{Sig},\mathcal{A}}(n) = 1\right]$$

We say that a signature scheme $\mathcal{S}$ *is* EUF-CMA secure*, if for any* $\mathcal{A}$ *the advantage function is negligible (as a function in* n*).*

The definitions for certification schemes work analogously. That is, a certification scheme consists of three algorithms $\mathcal{C} = (\mathsf{CKG}, \mathsf{CA}, \mathsf{CVf})$ for creating the authority's key pair, the certification of a public key, and for verifying a public key with respect to a certificate. We allow for multiple certifications of the same public key but assume that each certification requests is accompanied by an identifier id which will be included in *certID*. Then we can define unforgeability as for signatures, implying that the adversary cannot forge a valid certificate for a new public key or for a previously certified key under a new identity. We write $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{C},\mathcal{A}}$ for the advantage of an adversary in the $\mathsf{EUF\text{-}CMA}$ game against a certification scheme. In the EAC protocol the authority's public key is given by pk_{CVCA} and the key generation, certificate creation and certificate verification are often described implicitly only.

Furthermore, we can define message authentication codes (MACs) $\mathcal{M} = (\mathsf{MKG}, \mathsf{MAC}, \mathsf{MVf})$ analogously, except that the key generation algorithm only outputs a single secret key and the adversary does not receive any initial input in the attack. We write $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{M},\mathcal{A}}$ for the advantage of an adversary $\mathcal{A}$ in this game.

Finally, we need that the compression function Compr is collision-resistant. That is, for an adversary $\mathcal{A}$ it should be infeasible to find group elements $X \neq Y$ such that $\mathsf{Compr}(X) = \mathsf{Compr}(Y)$. We write $\mathsf{Adv}^{\mathsf{CR}}_{\mathsf{Compr},\mathcal{A}}$ to denote the advantage of such an adversary $\mathcal{A}$. We remark that we actually need a weaker requirement from Compr, resembling second preimage resistance, namely that for a random group element X it should be hard to find a colliding different Y, when given the discrete logarithm of X with respect to the group.

Next, we define our version of the $\mathsf{PRF\text{-}ODH}$ assumption as a slight extension to the original definition given in [25,26]. In accordance with the systematic study of the $\mathsf{PRF\text{-}ODH}$ assumption by Brendel et al. [7], we term our notion $\mathsf{mmPRF\text{-}ODH}$, which corresponds to the strongest variant with multiple queries to both ODH oracles.

Definition 4 ($\mathsf{mmPRF\text{-}ODH}$ assumption). *Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order q with generator g, and let $\mathsf{PRF}\colon \mathbb{G} \times \{0,1\}^* \to \{0,1\}^n$ be a pseudo-random function with keys $K \in \mathbb{G}$, input strings $x \in \{0,1\}^*$, and output strings $y \in \{0,1\}^n$, i.e., $y \leftarrow \mathsf{PRF}(K, x)$.*

We define the following $\mathsf{mmPRF\text{-}ODH}$ security game $G^{\mathsf{mmPRF\text{-}ODH}}_{\mathsf{PRF},\mathcal{A}}$ between a challenger $\mathcal{C}$ and a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$.:

Setup. *The challenger $\mathcal{C}$ samples $u \xleftarrow{\$} \mathbb{Z}_q$ and provides $\mathbb{G}, g$, and g^u to the adversary $\mathcal{A}$.*

Query Phase 1. *$\mathcal{A}$ can issue arbitrarily many queries to the following oracle ODH_u.*

ODH_u **oracle.** *On a query of the form (A, x), the challenger first checks if $A \notin \mathbb{G}$ and returns $\perp$ if this is the case.*
Otherwise, it computes $y \leftarrow \mathsf{PRF}(A^u, x)$ and returns y.

Challenge. *Eventually, $\mathcal{A}$ issues a challenge query $x^\star$. On this query, $\mathcal{C}$ samples $v \xleftarrow{\$} \mathbb{Z}_q$ and a bit $b \xleftarrow{\$} \{0,1\}$ uniformly at random. It then computes*

$y_0^\star = \mathsf{PRF}(g^{uv}, x^\star)$ and samples $y_1^\star \xleftarrow{\$} \{0,1\}^n$ uniformly random. The challenger returns $(g^v, y_b^\star)$ to $\mathcal{A}$.

Query Phase 2. *Next, $\mathcal{A}$ may issue (arbitrarily many and interleaved) queries to the following oracles ODH_u and ODH_v.*

ODH_u **oracle.** *On a query of the form (A, x), the challenger first checks if $A \notin \mathbb{G}$ or $(A, x) = (g^v, x^\star)$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{PRF}(A^u, x)$ and returns y.*

ODH_v **oracle.** *On a query of the form (B, x), the challenger first checks if $B \notin \mathbb{G}$ or $(B, x) = (g^u, x^\star)$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{PRF}(B^v, x)$ and returns y.*

Guess. *Eventually, $\mathcal{A}$ stops and outputs a bit b'.*

We say that the adversary wins the mmPRF-ODH *game if $b' = b$ and define the advantage function*

$$\mathsf{Adv}_{\mathsf{PRF},\mathcal{A}}^{\mathsf{mmPRF\text{-}ODH},\mathbb{G}}(n) := 2 \cdot \left(\Pr[b' = b] - \frac{1}{2} \right)$$

and, assuming a sequence of groups in dependency of the security parameter, we say that a pseudorandom function PRF *with keys from $(\mathbb{G}_n)_n$ provides* mmPRF-ODH *security if for any $\mathcal{A}$ the advantage $\mathsf{Adv}_{\mathsf{PRF},\mathcal{A}}^{\mathsf{mmPRF\text{-}ODH}}(n)$ is negligible in the security parameter n.*

3.4 Analysis

Under the assumptions described above we can show that the EAC+0RTT protocol satisfies the required security properties:

Theorem 1. *The EAC+0RTT protocol is* Match*-secure. For any efficient adversary $\mathcal{A}$ we have*

$$\mathsf{Adv}_{EAC,\mathcal{A}}^{\mathsf{Match}} \leq q_p^2 \cdot \min\{2^{-|\mathsf{nonce}|}, \tfrac{1}{q}\}$$

where q_p is the maximum number of sub protocol executions, $|\mathsf{nonce}|$ is the bit-length of each of the nonces r_C, r'_C, r''_C, and q is the order of the group from which (ephemeral) keys are chosen.

Similarly, we can show key secrecy, and even argue forward secrecy with respect to subsequent terminal corruptions. We note that forward secrecy with respect to chip corruptions is impossible to achieve for EAC since the chip does not generate ephemeral keys for executions but rather uses the long-term secrets:

Theorem 2. *The EAC+0RTT protocol provides key secrecy (with* responder *forward secrecy). That is, for any efficient adversary $\mathcal{A}$ there exist efficient adversaries $\mathcal{B}_3, \mathcal{B}_4, \mathcal{B}_5, \mathcal{B}_{10/11}$ such that*

$$\begin{aligned}\mathsf{Adv}_{\mathsf{KE},\mathcal{A}}^{\mathsf{BR},\mathcal{D}}(n) \leq\ & 3q_p^2 \cdot \max\{2^{-|\mathsf{nonce}|}, \tfrac{1}{q}\} + \mathsf{Adv}_{\mathsf{Compr},\mathcal{B}_3}^{\mathsf{CR}} \\ & + \mathsf{Adv}_{\mathcal{C},\mathcal{B}_4}^{\mathsf{EUF\text{-}CMA}} + q_T \cdot \mathsf{Adv}_{\mathcal{S},\mathcal{B}_5}^{\mathsf{EUF\text{-}CMA}} \\ & + 4q_p \cdot q_C \cdot \max\{q_p, q_{\mathsf{sskid}}\} \cdot \mathsf{Adv}_{\mathcal{B}_{10/11}}^{\mathsf{mmPRF\text{-}ODH}}\end{aligned}$$

where q_p is the maximum number of sub protocol executions, q_s is the maximal number of sessions, q_C is the maximal number of chips, q_T is the maximal number of terminals, $|\mathsf{nonce}|$ is the bit-length of each of the nonces r_C, r'_C, r''_C, and q is the order of the group from which (ephemeral) keys are chosen.

Remark 1. It may come as a surprise that the unforgeability of the MAC does not enter the security bound. This is due to the fact that we are "only" interested in key secrecy in the above theorem, stating that *at most* the intended partner can compute the session key and that seeing other session keys does not facilitate this task. The former is ensured by the certification of the chip's long-term key and the fact that one cannot corrupt the chip. The latter is already captured by the $\mathsf{mmPRF\text{-}ODH}$ assumption, which states that learning related values of the PRF does not help to distinguish the challenge value from random.

Remark 2. Note that our analysis does not provide any form of key confirmation nor entity authentication. In fact, the final MAC can be seen as providing exactly these properties [18].

4 Variations

There exist several alternatives to implement 0RTT executions. For example, the 0RTT keys may be established either in the fashion of a Diffie-Hellman key exchange or—forgoing forward secrecy— rather from pre-shared keys (derived as additional key material in the previous round). It is also interesting to investigate different ways of handling negotiation failures in the 0RTT case. In the following, we therefore present different choices for the 0RTT flow.

4.1 Diffie-Hellman Variant

The 0RTT EAC extension presented in Sect. 2.2 is based on a Diffie-Hellman style key agreement. Similar implementations can also be found in Google's QUIC protocol and in earlier draft versions of TLS 1.3 (draft 12 [30] and earlier).

4.2 Pre-shared Key Variant

From draft 13 [31] onward, TLS 1.3 replaces the DH-based variant of 0RTT handshakes by a pre-shared key (PSK) alternative. The pre-shared key is established either out of band or, more commonly, in a preceding interaction between server and client. Once a full handshake has been completed, the client receives a so-called PSK identity from the server. The PSK was derived in the initial handshake and can then be used by the client to derive keys for future (0RTT) handshakes. To initiate a 0RTT handshake, the client simply incorporates the `early_data` and `pre_shared_key` extension in the `ClientHello`, followed by the application data. After the successful processing of the data, the server then responds with the `ServerHello` and a forward-secret key is then derived as in the ordinary handshake.

In principle, one could also imagine a similar approach for the EAC protocol, using the pre-shared keying material instead of the shared Diffie-Hellman key. Note, however, that this may require further changes to the EAC protocol (for the additional keying material) and that, unlike the Diffie-Hellman version, this does not provide any (terminal) forward secrecy.

4.3 Error Handling

Zero round-trip time may not be supported by all servers, or there may occur errors in trying to decrypt the early data. Here we discuss how such problems are dealt with in other settings, and how one can proceed in the EAC case.

Google's QUIC Protocol. From a design perspective, all handshakes in QUIC are also 0RTT handshakes, of which some may fail. The server replies with a `ServerHello` if all necessary information to complete the handshake was contained in the preceding `ClientHello`. If this was not the case, the server sends a rejection message encompassing information that allows the client to make progress in a next handshake attempt. The type and extent of information sent along with the rejection message can be chosen individually by the server but must not prevent clients from establishing a valid handshake within a reasonable time frame.

TLS 1.3 Draft 20. Upon receiving a 0RTT handshake request with encrypted early data, the server can answer in three ways: It may either disregard the 0RTT extension and return no response, causing the client to fall back to the standard 1RTT handshake. Or it may return the empty extension, thereby signalling to the client that prior validation checks were successful and that the server intends to process the received early data. Furthermore, the server may send a `HelloRetryRequest` to the client asking it to send a `ClientHello` without the `early_data` extension.

0RTT EAC. In case of failure, we expected the client to fall back to a full EAC protocol execution consisting of terminal and chip authentication. This may seem like an expensive step in view of performance, especially if the semi-static key used by the client is simply outdated. If the terminal does not support 0RTT, fall back to full EAC is clearly inevitable.

Furthermore, we emphasize that it is in general not possible for terminals to identify outdated keys. In order for a terminal to detect this (i.e., to distinguish unknown keys from outdated keys), it must keep at least the last used value of pk_T^{semi} when updating to a new value $pk_T^{\mathsf{semi}'}$. Keeping state is commonly seen as not recommendable, if not infeasible, in most use cases. However, we note that a chip receives all the data it needs to initiate future 0RTT handshakes with a 0RTT-supporting terminal during the terminal authentication phase of the EAC protocol. Therefore, it is sufficient for the chip to carry out the TA phase before the 0RTT handshake can be re-tried. In light of this, it is also conceivable for terminals to proceed similarly to the mechanism deployed in the

QUIC protocol and to reply with the current authenticated semi-static key, i.e., to send $cert_T, pk_T^{\mathsf{semi}}, s_T$ where $s_T \leftarrow \mathsf{Sig}(sk_T, pk_T^{\mathsf{semi}})$.

5 Conclusion

The Extended Access Control (EAC) protocol is a universal solution for key establishment between two parties. In this work, we presented a 0RTT mode for the EAC protocol which allows to reduce the latency of recurring connections. It is noteworthy that this 0RTT mode can be added as an extension with minimal changes to the original protocol. We further showed that EAC+0RTT can be proven secure in the multi-stage setting of the Bellare-Rogaway model. Thus, the modified protocol still achieves the common security properties of an authenticated key exchange protocol.

Acknowledgements. We thank the anonymous reviewers for valuable comments. This work has been co-funded by the DFG as part of project D.2 within the RTG 2050 "Privacy and Trust for Mobile Users", as well as part of project S4 within the CRC 1119 CROSSING.

References

1. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (Aug (1994)
2. Bender, J., Dagdelen, Ö., Fischlin, M., Kügler, D.: Domain-specific pseudonymous signatures for the german identity card. In: Gollmann, D., Freiling, F.C. (eds.) ISC 2012. LNCS, vol. 7483, pp. 104–119. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33383-5_7
3. Bender, J., Dagdelen, Ö., Fischlin, M., Kügler, D.: The PACE—AA Protocol for Machine Readable Travel Documents, and Its Security. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 344–358. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32946-3_25
4. Bender, J., Fischlin, M., Kügler, D.: Security analysis of the PACE key-agreement protocol. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) ISC 2009. LNCS, vol. 5735, pp. 33–48. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04474-8_3
5. Bender, J., Fischlin, M., Kügler, D.: The PACE|CA protocol for machine readable travel documents. In: Bloem, R., Lipp, P. (eds.) INTRUST 2013. LNCS, vol. 8292, pp. 17–35. Springer, Cham (2013). doi:10.1007/978-3-319-03491-1_2
6. Brendel, J., Fischlin, M.: Zero Round-Trip Time for the Extended Access Control Protocol. Cryptology ePrint Archive, Report 2017/060 (2017). http://eprint.iacr.org/2017/060
7. Brendel, J., Fischlin, M., Günther, F., Janson, C.: PRF-ODH: Relations, Instantiations, and Impossibility Results. Cryptology ePrint Archive, Report 2017/517 (2017). http://eprint.iacr.org/2017/517
8. Brzuska, C.: On the foundations of key exchange. Ph.D. thesis, Technische Universität Darmstadt, Darmstadt, Germany (2013). http://tuprints.ulb.tu-darmstadt.de/3414/

9. Brzuska, C., Fischlin, M., Warinschi, B., Williams, S.C.: Composability of Bellare-Rogaway key exchange protocols. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM CCS 2011, pp. 51–62. ACM Press, October 2011
10. BSI (Bundesamt für Sicherheit in der Informationstechnik, Federal Office for Information Security): Technical Guideline TR-03110: Advanced Security Mechanisms for Machine Readable Travel Documents: Extended Access Control (EAC), Password Authenticated Connection Establishment (PACE), and Restricted Identification (RI. BSI-TR-03110, version 2.0) (2008)
11. Cheng, Y., Chu, J., Radhakrishnan, S., Jain, A.: TCP Fast Open, RFC 7413, Internet Engineering Task Force (IETF), December 2014
12. Cooper, D., Ferraiolo, H., Mehta, K., Francomacaro, S., Chandramouli, R., Mohler, J.: Interfaces for Personal Identity Verification - Part 1: PIV Card Application Namespace, Data Model and Representation, May 2015
13. Coron, J.-S., Gouget, A., Icart, T., Paillier, P.: Supplemental access control (PACE v2): security analysis of PACE integrated mapping. In: Naccache, D. (ed.) Cryptography and Security: From Theory to Applications. LNCS, vol. 6805, pp. 207–232. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28368-0_15
14. Dagdelen, Ö., Fischlin, M.: Security analysis of the extended access control protocol for machine readable travel documents. In: Burmester, M., Tsudik, G., Magliveras, S., Ilić, I. (eds.) ISC 2010. LNCS, vol. 6531, pp. 54–68. Springer, Heidelberg (2011). doi:10.1007/978-3-642-18178-8_6
15. Dagdelen, Ö., Fischlin, M., Gagliardoni, T., Marson, G.A., Mittelbach, A., Onete, C.: A cryptographic analysis of OPACITY - (extended abstract). In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 345–362. Springer, Heidelberg (2013)
16. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015, pp. 1197–1210. ACM Press, October 2015
17. Fischlin, M., Günther, F.: Multi-stage key exchange and the case of Google's QUIC protocol. In: Ahn, G.J., Yung, M., Li, N. (eds.) ACM CCS 2014, pp. 1193–1204. ACM Press, November 2014
18. Fischlin, M., Günther, F., Schmidt, B., Warinschi, B.: Key confirmation in key exchange: a formal treatment and implications for TLS 1.3. In: 2016 IEEE Symposium on Security and Privacy, pp. 452–469. IEEE Computer Society Press, May 2016
19. Gilson, B., Baldridge, T.: PKI (CAK) – Enabled PACS with PIV Card: PACS Lessons Learned and Need for Speed, May 2015. Presentation at FIPS 201–2 Supporting Special Publications Workshop. http://csrc.nist.gov/groups/SNS/piv/fips_201-2_march_2015/day_one/gilson_baldridge_piv-cak_enabled_pacs_fips201-2_2015.pdf
20. Google: QUIC, a multiplexed stream transport over UDP (2016). https://www.chromium.org/quic
21. Hale, B., Jager, T., Lauer, S., Schwenk, J.: Speeding: on low-latency key exchange. Cryptology ePrint Archive, Report 2015/1214 (2015). http://eprint.iacr.org/2015/1214
22. Hanzlik, L., Krzywiecki, Ł., Kutyłowski, M.: Simplified PACE|AA protocol. In: Deng, R.H., Feng, T. (eds.) ISPEC 2013. LNCS, vol. 7863, pp. 218–232. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38033-4_16
23. Hanzlik, L., Kutyłowski, M.: Restricted identification secure in the extended Canetti-Krawczyk model. J. Univ. Comput. Sci. **21**(3), 419–439 (2015)

24. ICAO: Machine Readable Travel Documents, Part 11, Security Mechanisms for MRTDs. Doc 9303, 7th edn. (2015)
25. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 273–293. Springer, Heidelberg (2012)
26. Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the TLS protocol: a systematic analysis. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 429–448. Springer, Heidelberg (2013)
27. Kutyłowski, M., Krzywiecki, Ł., Kubiak, P., Koza, M.: Restricted identification scheme and Diffie-Hellman linking problem. In: Chen, L., Yung, M., Zhu, L. (eds.) INTRUST 2011. LNCS, vol. 7222, pp. 221–238. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32298-3_15
28. Morgner, F., Bastian, P., Fischlin, M.: Attribute-based access control architectures with the eIDAS protocols. In: SSR 2016: Security Standardisation Research. LNCS, vol. 10074, pp. 205-226. Springer, Heidelberg (2016). doi:10.1007/978-3-319-49100-4_9
29. Morgner, F., Bastian, P., Fischlin, M.: Securing transactions with the eIDAS protocols. In: Foresti, S., Lopez, J. (eds.) WISTP 2016. LNCS, vol. 9895, pp. 3–18. Springer, Cham (2016). doi:10.1007/978-3-319-45931-8_1
30. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3, draft-ietf-tls-tls13-12. https://tools.ietf.org/html/draft-ietf-tls-tls13-12
31. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3, draft-ietf-tls-tls13-13. https://tools.ietf.org/html/draft-ietf-tls-tls13-13
32. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3 - draft-ietf-tls-tls13-20. https://tools.ietf.org/html/draft-ietf-tls-tls13-20
33. Smart Card Alliance: Industry Technical Contributions: OPACITY. http://www.smartcardalliance.org/smart-cards-contributions-opacity/