Symbolic and Computational Mechanized Verification of the ARINC823 Avionic Protocols

Bruno Blanchet

Symbolic and Computational Mechanized Verification
of the ARINC823 Avionic Protocols

Bruno Blanchet
Inria Paris, France
Email: bruno.blanchet@inria.fr

Abstract—We present the first formal analysis of two avionic protocols that aim to secure air-ground communications, the ARINC823 public-key and shared-key protocols. We verify these protocols both in the symbolic model of cryptography, using ProVerif, and in the computational model, using CryptoVerif. While we confirm many security properties of these protocols, we also find several weaknesses, attacks, and imprecisions in the standard. We propose fixes for these problems. This case study required the specification of new cryptographic primitives in CryptoVerif. It also illustrates the complementarity between symbolic and computational verification.

I. Introduction

Securing electronic communications between aircrafts and ground entities (control towers, airlines) becomes more and more important. Indeed, such electronic communications convey more and more important information and aircrafts are prominent targets for attacks, such as terrorist attacks. So, even if most air-ground communications are currently sent in the clear, the adoption of secured communications seems unavoidable.

Several standards have been developed [2], [34], [35] or are in development (third edition of [34]) in order to secure these communications. In this paper, we study the ARINC823 standard [2]. This standard aims to secure ACARS (Aircraft Communications Addressing and Reporting System) messages. These messages are short text messages, which include air traffic control messages, such as clearance messages, flight plans, weather information, as well as maintenance messages, so that needed maintenance operations can be planed at the next airport.

In addition to security (authentication and secrecy of the messages), other goals are apparent in the design of the standard:

- Bandwidth: the available bandwidth is limited and the ACARS messages are relatively short, so the protocol is designed to minimize the additional bandwidth required for security.
- Resistance to failures: the messages are conveyed even in case of failure of the security system.
- Flexibility: the protocol is designed to provide different levels of security: authentication and secrecy, authentication only, and no security, on a message-per-message basis.

These additional goals justify the design of specific protocols. The ARINC823 standard provides two such protocols:

a public-key protocol and a shared-key protocol. (The latter assumes that the aircraft and the ground entity share a long-term secret key.) To our knowledge, the public-key protocol has been implemented and is currently used by one major airline. We do not know of a current usage of the shared-key protocol, but it might be used in the future.

Other avionic protocols share the constraints of low bandwidth and resistance to failures, but some of them are much simpler. For instance, the third edition of [34] just aims at authentication and does not require flexibility, so its analysis does not need to consider as many cases as the one of the ARINC823 standard.

In order to obtain guarantees that the ARINC823 protocols are secure, we use mechanized protocol verification tools. These tools typically rely on one of the following two models of cryptography:

- The *symbolic model*, so-called Dolev-Yao model [31], [39], is an abstract model in which cryptographic primitives are considered as perfect black-boxes, messages are terms on these primitives, and the adversary is restricted to apply only these primitives. This model facilitates formal verification, and many automatic protocol verifiers (AVISPA [3], F7 [12], FDR [37], ProVerif [21], Scyther [28], Tamarin [45], ...) exist in this model.
- The *computational model* [48], typically used for manual cryptographic proofs, is more concrete. In this model, messages are bitstrings, cryptographic primitives are functions on these bitstrings, and the adversary is any probabilistic Turing machine. Formal verification is more difficult in this model, and most proofs are manual. Some mechanized verifiers exist (CryptoVerif [17], [18], CertiCrypt [7], [8] and its successor EasyCrypt [4], [6], Computational F7 [15] and its successors F* and RF* [5]).

We consider both models in our study, and rely on ProVerif (http://proverif.inria.fr) for the symbolic analysis and on CryptoVerif (http://cryptoverif.inria.fr) for the computational analysis. Our analysis does not consider side-channels (such as timing and power consumption), and works on a model of the protocol, not on an implementation. Our goal is to verify the ARINC823 standard, not a particular implementation (to which we did not have access). Having a secure standard is an essential first step, even if additional weaknesses may appear at the implementation level.

Contributions: We formally model both the public-key (Section II) and the shared-key (Section III) ARINC823

protocols, in ProVerif and CryptoVerif. Our models are available at http://prosecco.inria.fr/personal/bblanche/arinc823/ We study many security properties of these protocols: freshness of messages, absence of replays, authentication, secrecy, forward secrecy, resistance to key compromise impersonation attacks. While we confirm important properties (the public-key protocol guarantees secrecy and authentication of the application data messages), we also find several weaknesses and imprecisions in the standard, in particular related to the resistance to failures and to the protection against replays. In the shared-key protocol, a subtle, computational replay attack may lead to the breach of secrecy. We propose fixes for these weaknesses, and prove their security. Moreover, the protocols use undesirable cryptographic constructions; in particular, they share keys between several primitives. The public-key protocol does not satisfy forward secrecy and is subject to key compromise impersonation attacks. While these security properties may not be explicit requirements of the standard, the protocol can be strengthened to guarantee them and to avoid key sharing between primitives. We propose such a strengthened protocol and verify it.

We reported our results to Honeywell, the industry editor for and a key technical contributor to the ARINC823 standard. Honeywell gave us helpful feedback on our analysis of the public-key protocol, which they implement. We summarize this feedback in the rest of this paper.

This case study required the specification of new cryptographic primitives in CryptoVerif: a joint assumption between gap Diffie-Hellman and signatures (Section II-C 2)), and a counter-mode encryption scheme (Section II-C 5)). As we shall see, this case study also illustrates the complementarity between verification in the symbolic and in the computational models and the usage of manual arguments in order to complement some ProVerif proofs.

Related Work: Other approaches to protocol verification have already been discussed above. Many other cases studies have been conducted using ProVerif (see [16] for references), a few using CryptoVerif [1], [14], [20], [23], [25], [46]. This study is one of the rare that combine both tools [13], [36]: [36] studies the Signal messaging protocol and [13] studies TLS 1.3 Draft-18. Those two studies generate ProVerif scripts from JavaScript code for symbolic verification, and additionally use manually written CryptoVerif scripts to obtain computational guarantees. This study is probably the one that exploits most the complementarity between these tools. We do not know of any previous formal verification of the ARINC823 protocols.

II. AMS PROTOCOL USING PUBLIC/PRIVATE KEYS

In this section, we study the AMS (ACARS Message Security) protocol using public/private keys as defined in [2, Attachment 7].

A. Protocol Description

Let us first provide a simplified description of the protocol; our models cover the full specification. The protocol
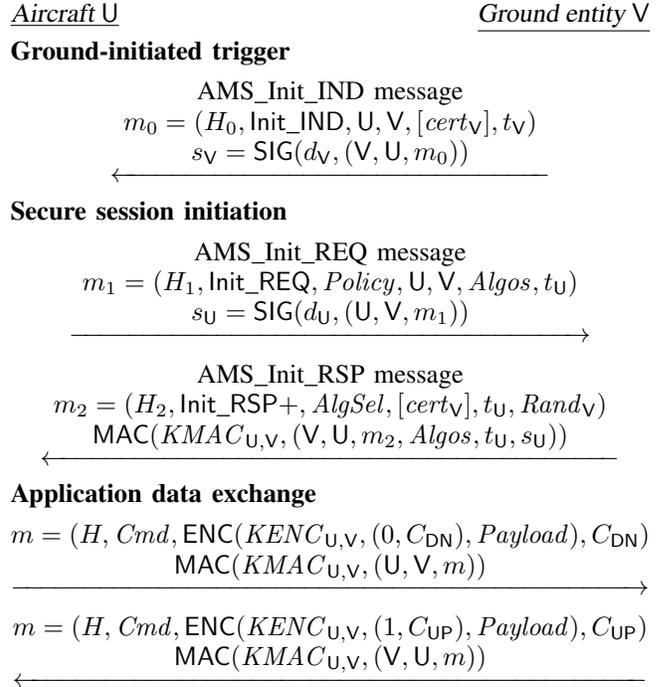

Aircraft U *Ground entity* V

Ground-initiated trigger

$$\text{AMS\_Init\_IND message}$$
$$m_0 = (H_0, \mathsf{Init\_IND}, \mathsf{U}, \mathsf{V}, [cert_\mathsf{V}], t_\mathsf{V})$$
$$s_\mathsf{V} = \mathsf{SIG}(d_\mathsf{V}, (\mathsf{V}, \mathsf{U}, m_0))$$
$\longleftarrow$

Secure session initiation

$$\text{AMS\_Init\_REQ message}$$
$$m_1 = (H_1, \mathsf{Init\_REQ}, Policy, \mathsf{U}, \mathsf{V}, Algos, t_\mathsf{U})$$
$$s_\mathsf{U} = \mathsf{SIG}(d_\mathsf{U}, (\mathsf{U}, \mathsf{V}, m_1))$$
$\longrightarrow$

$$\text{AMS\_Init\_RSP message}$$
$$m_2 = (H_2, \mathsf{Init\_RSP+}, AlgSel, [cert_\mathsf{V}], t_\mathsf{U}, Rand_\mathsf{V})$$
$$\mathsf{MAC}(KMAC_{\mathsf{U},\mathsf{V}}, (\mathsf{V}, \mathsf{U}, m_2, Algos, t_\mathsf{U}, s_\mathsf{U}))$$
$\longleftarrow$

Application data exchange

$$m = (H, Cmd, \mathsf{ENC}(KENC_{\mathsf{U},\mathsf{V}}, (0, C_\mathsf{DN}), Payload), C_\mathsf{DN})$$
$$\mathsf{MAC}(KMAC_{\mathsf{U},\mathsf{V}}, (\mathsf{U}, \mathsf{V}, m))$$
$\longrightarrow$

$$m = (H, Cmd, \mathsf{ENC}(KENC_{\mathsf{U},\mathsf{V}}, (1, C_\mathsf{UP}), Payload), C_\mathsf{UP})$$
$$\mathsf{MAC}(KMAC_{\mathsf{U},\mathsf{V}}, (\mathsf{V}, \mathsf{U}, m))$$
$\longleftarrow$

Figure 1. Simplified AMS protocol using public/private keys

---

establishes a secure session between an aircraft U and a ground entity V, as shown in Figure 1. The aircraft initially has an elliptic curve key pair (secret key d_U and public key Q_U) and similarly the ground entity has an elliptic curve key pair (secret key d_V and public key Q_V). The ground entity may request that the aircraft initiates a secure session. In this case, it sends a ground-initiated trigger, AMS_Init_IND, which consists of a header field H_0, a constant command $\mathsf{Init\_IND}$ for initiation indication, the identities of the aircraft U and ground entity V, optionally the certificate $cert_\mathsf{V}$ for the public key Q_V of V (the certificate must be present if the aircraft does not already have it), and a timestamp t_V. This message is signed with the secret key d_V. Upon receipt of this message, the aircraft verifies its own identity, the certificate if present, the signature, and verifies that the timestamp t_V is recent enough:

$$t - 60 \text{ seconds} \leq t_\mathsf{V} \leq t + 120 \text{ seconds}.$$

where t is the current time at the aircraft. Then, it initiates a secure session. The aircraft may also directly initiate a secure session; the ground-initiated trigger is then omitted.

To initiate a secure session, the aircraft sends an AMS_Init_REQ message, which consists of a header field H_1, a constant command $\mathsf{Init\_REQ}$ for initiation request, an identifier $Policy$ for a security policy that determines when messages are encrypted and/or MACed, the identities of the aircraft U and ground entity V, the algorithms $Algos$ that the aircraft supports (encryption, MAC, compression), and a timestamp t_U. This message is signed with the secret key d_U. Upon receipt of this message, the ground entity

V verifies its own identity, gets the aircraft certificate for the public key Q_U from its local storage or from a PKI certificate distribution service, verifies the signature, verifies that $(Algos, t_U, s_U)$ is unique for the aircraft initiating a new session, and verifies that the timestamp t_U is recent enough:

$$t - 60 \text{ seconds} \leq t_U \leq t + 120 \text{ seconds.} \quad (1)$$

where t is the current time at the ground entity. It selects supported algorithms among those proposed by the aircraft, and stores them in $AlgSel$, computes a shared Diffie-Hellman secret $Z_{U,V} = d_V Q_U$, chooses a 32-bit random value $Rand_V$, and computes the following values:

$$X_{U,V} = \mathsf{HASH}((Algos, t_U, s_U, Rand_V))$$
$$KMAC_{U,V} = \mathsf{KDF}(Z_{U,V}; m; \mathsf{concatSI}(1, MAC\_algo,$$
$$X_{U,V}, U, V, m))$$
$$KENC_{U,V} = \mathsf{KDF}(Z_{U,V}; n; \mathsf{concatSI}(2, ENC\_algo,$$
$$X_{U,V}, U, V, n))$$

where the hash function HASH is SHA256, MAC_algo is the selected MAC algorithm and m is the length of its keys, ENC_algo is the selected encryption algorithm and n is the length of its keys, the function concatSI concatenates its arguments, the key derivation function KDF is defined by ANSI X9.63 but uses SHA256 instead of SHA1 as underlying hash function, and $\mathsf{KDF}(Z; m; X)$ returns an m-bit key.

Then the ground entity sends the AMS_Init_RSP message, which consists of a header field H_2, a constant command Init_RSP+ for a positive initiation response (a Init_RSP− message may be sent when the ground entity rejects the secure session), the selected algorithms $AlgSel$, optionally the certificate $cert_V$ for the public key Q_V of V (the certificate must be present if the aircraft does not already have it), the timestamp t_U, and the random value $Rand_V$. This message is MACed together with the fields $Algos, t_U, s_U$ received in the AMS_Init_REQ message, using the key $KMAC_{U,V}$.

Upon receipt of this message, the aircraft verifies that the received timestamp t_U is equal to the one it sent in the AMS_Init_REQ message, verifies the certificate if present, computes the shared Diffie-Hellman secret $Z_{U,V} = d_U Q_V$, computes $X_{U,V}$, $KMAC_{U,V}$, and $KENC_{U,V}$ as above, and verifies the MAC.

When all these steps succeed, the aircraft and ground entity can exchange application data messages in an encrypt-then-MAC tunnel, using $KENC_{U,V}$ as encryption key and $KMAC_{U,V}$ as MAC key. They use two distinct counters C_{DN} and C_{UP}, one in each direction, to order the messages and to prevent replays. These counters are incremented when a message is sent or received in the considered direction.

The specified MAC algorithm is HMAC-SHA256, truncated to 32, 64, or 128 bits. The specified encryption algorithm is AES128-CFB128. The specified compression algorithms are Dynamic Markov Compression and DEFLATE. These algorithms are chosen with bandwidth in mind: the

MACs are rather short, the encryption algorithm is such that the length of the ciphertext is equal to the length of the plaintext (at the bit level), and compression may be used. The protocol allows specifying additional algorithms in the future, which are negotiated via the fields $Algos$ and $AlgSel$. The MAC and encryption algorithms are chosen at the beginning of the protocol. The compression algorithm may vary for each message, among the supported algorithms. The chosen compression algorithm is included in the header field H of each message. When compression is active, the payload is compressed before encryption.

The protocol offers 4 protection modes for the messages:
- BOTH: the message is encrypted then MACed.
- AUTH: the message is just MACed.
- NONE: no encryption nor MAC.
- SIGN: the message is signed.

Protection mode SIGN is used only in the AMS_Init_IND and AMS_Init_REQ messages. The AMS_Init_RSP message always uses protection mode AUTH. The application data messages may use the modes BOTH (as represented in Figure 1), AUTH, or NONE. The protection mode of each message is included in the header field H. It is chosen based on the security policy of identifier $Policy$ and the type of message sent. Finally, the aircraft may or may not support encryption. In the latter case, messages are never encrypted, even in protection mode BOTH.

B. Modeling Notes

Our models of this protocol have the following features:

1) Sessions. The models consider a single honest aircraft U and a single honest ground entity V. Other aircrafts and ground entities are included in the adversary. The aircraft U is willing to run the protocol with any ground entity, and the ground entity V is willing to run the protocol with any aircraft. We prove security properties for the sessions between the honest aircraft U and ground entity V.

The public-key protocol and the shared-key protocol do not share secrets, so we can consider them separately without missing attacks: the shared-key protocol is supposed to be included in the adversary when we analyze the public-key protocol.

The models consider an unbounded number of executions of the protocol.

2) Algorithms. The models consider only the algorithms currently specified for compression, encryption, and MAC. We do not model any algorithms reserved for future use. In the future, it might be interesting to verify that the algorithm negotiation remains secure even if algorithms are added.

3) Certificates. The certificates are modeled in a fairly abstract way: the certificate that Q_U is the public key of U is modeled as a signature of the pair (U, Q_U) under the secret key sCA of a certificate authority.

4) Equations for encoding and compression. To model that encoding and compression are the identity when they are OFF, ProVerif uses equations $\mathsf{encode}(\mathsf{encode\_OFF}, x) = x$ and $\mathsf{compress}(\mathsf{comp\_OFF}, x) = x$. These equations considerably increase the runtime and memory consumption of

ProVerif, to the point that ProVerif could not prove authentication for the public-key protocol with these equations. Indeed, with these equations, ProVerif needs to consider two cases for each encode operation: either the mode is encode_OFF, and the encode operation is the identity, or it is another mode, and encode is a non-identity function, considered as a free function symbol. The same holds for compress and for the reverse operations, so the number of cases to consider doubles for each encode, decode, compress, and decompress. Therefore, we omit these equations, to avoid such case distinctions. CryptoVerif proves desired properties for any injective encoding and compression functions, in particular those that satisfy these equations.

5) Timestamps. The specification [2, Table 7.2.1-1, Step N04, Item 3] says that a timestamp t_U is accepted at time t when (1) holds. We found this inequality surprising: if we suppose that the message may take between 0 and 60 seconds to go from one entity to the other, and the clocks can differ by at most 60 seconds, we obtain

$$t - 120 \text{ seconds} \leq t_U \leq t + 60 \text{ seconds} \qquad (2)$$

rather than the inequality (1). Honeywell confirmed that there is a typo in the standard and that (2) is the correct inequality. We rely on the corrected inequality in the rest of this document.

We use the following abstract model of timestamps. We define a channel timech on which we input the current time, and a predicate check_time such that $\text{check\_time}(t_U, t)$ is true if and only if (2). In the models, since ProVerif and CryptoVerif cannot model time precisely, the channel timech is in fact a public channel, so the adversary can send anything on it, and the definition of the predicate check_time is left unspecified, so the proofs done by ProVerif and CryptoVerif are valid for any definition of check_time.

6) Replay protection. We do not model the check mentioned in [2, Table 7.2.1-1, Step N07, Item 4]:

> [The ground entity] verifies that the received AMS_Appendix field [$Algos, t_U, s_U$ in Figure 1] is unique for the aircraft initiating a new session. Note 1—This check is necessary to ensure that the received AMS_Init_REQ message is associated with a new initiation request and not an ACARS retransmission of an AMS_Init_REQ message for which a secure session is already established.

Above, the AMS_Appendix field is considered non-unique when a session with the aircraft is established with that appendix [2, Table 7.2.1-2, Step N07, Case 2].

Even though replays of AMS_Init_REQ messages may not be problematic, we clarify that, in order to avoid all such replays, it is not enough to verify that the received AMS_Appendix field does not occur in the AMS_Init_REQ messages of currently established sessions. With this check, in case a session terminates at a time t such that $t - 120$ seconds $\leq t_U$, the adversary can replay the initial AMS_Init_REQ message just after the session terminates, and this message will be accepted. To avoid all replays, the check should be defined as follows.

For each aircraft U, the ground entity should remember the AMS_Appendix field of the AMS_Init_REQ messages received from U with the most recent timestamp t_U, as long as t_U is such that $t - 120$ seconds $\leq t_U$ where t is the ground entity's local time. (Replays of AMS_Init_REQ messages older than 120 seconds are discarded by checking the timestamp.) A new AMS_Init_REQ message received from U should be accepted if and only if its timestamp t_U satisfies (2), and it is strictly more recent than the remembered timestamp t_U or it is equal to the remembered t_U and AMS_Appendix field is not among the remembered AMS_Appendix fields. (The AMS_Appendix field includes the signature s_U which is probabilistic, so each session uses a different signature, and hence a different AMS_Appendix field. A message with an AMS_Appendix field that has already been seen is therefore a replay.) This check guarantees that the ground entity never accepts two AMS_Init_REQ messages with the same U, s_U, t_U.

Moreover, in the case of a ground-initiated trigger, the uniqueness of the AMS_Appendix field is never checked. As a result, the adversary can replay the AMS_Init_IND message as long as its timestamp t_V satisfies the inequality $t - 120$ seconds $\leq t_V \leq t + 60$ seconds, where t is the aircraft entity's local time. In particular, if the adversary replays the AMS_Init_IND message while a secure session is established between the aircraft and the ground entity, this message will be accepted by the aircraft; according to [2, Table 7.2.2-1, Step G08, Item 2], this replay will cause the just started session to end and a new secure session to be initiated. If the entities handle a secure session restart by sending again the messages of the previous session, the attack may cause messages to be delayed by up to 120 seconds (perhaps using several replays of the first AMS_Init_IND message). It may also have more serious consequences if the entities forget the messages they cannot send in the initial session due to the unexpected termination. To avoid this attack, we recommend protecting against replays of the AMS_Init_IND message by using a check similar to the one mentioned above for the AMS_Init_REQ message. Honeywell agrees with the anti-replay checks we recommend.

7) Message counter. ProVerif and CryptoVerif cannot fully model the incrementation of the message counter and the tests performed on this counter to accept or reject the message. In ProVerif, the value of the counter is input on a public channel and the tests are omitted. The CryptoVerif model is more precise: it models that the sender never reuses the same counter in the same session, and that the receiver rejects a message when it has already seen a message with the same counter in the same session. It still omits that

highest received count $- 5 \leq$ *current received count*

$\leq$ *highest received count* $+ 5$

as mentioned in the message count windowing criteria [2, Table 7.3.1-1, Step D08]. (These windowing criteria allow a counter smaller than the highest received counter, to allow receiving out-of-order messages, which may happen when

the network is not entirely reliable.) The proved security properties hold a fortiori with this additional check.

8) Resistance to failures. [2, page 133, note 1] reads:

> If triggering secure session initiation is unsuccessful after an initial attempt and one re-attempt, then the ground entity assumes AMS is unavailable for the aircraft entity and sends queued messages in the clear.

Similarly, [2, page 130, note 1] reads:

> If secure session initiation is unsuccessful after an initial attempt and a configurable number of re-attempts, then the aircraft entity executes a configurable unavailable action, as described in Section 3.5.4.

and [2, Section 3.5.4] specifies

> Possible actions may include, but are not limited to, the following:
> - Attempt to send the message to an alternate AMS-equipped entity specified by the airline.
> - Send the message in the clear bypassing AMS.
> - Do not send the message.

Honeywell points out that, when an AMS-equipped aircraft entity is not able to establish a secure session with a ground entity, stakeholders (e.g., avionics suppliers, datalink service providers, airlines) that contributed to the development of the standard agreed that the ARINC 823 specification should include flexibility such that implementers may provide such customer-configurable actions.

This behavior may cause obvious security breaches when messages are sent in the clear: if an adversary manages to intercept or jam messages so that the secure session initiation fails even though AMS is actually available, then it will get the sent messages in the clear, thus breaking secrecy.

The intention of communicating the messages even in case of failure of the security system is understandable, but it conflicts with the goal of securing the communication: security is guaranteed only when a secure session is established. We recommend taking additional actions in case secure session initiation fails or messages are sent or received outside secure sessions, such as informing the pilot and reporting the problem to a computer maintenance team on the ground. We recommend mentioning that in the standard, even though Honeywell points out that user notifications and problem reporting are considered local implementation issues that are not within scope of the AMS interoperability specification. Our models include only the behavior inside secure sessions, so in our models, no message is sent in the clear when secure session initiation fails. With the migitations mentioned above, our security results still hold for the full protocol when no problem is reported to the pilot or to the ground maintenance.

9) Policy check. The models include the verification that messages were transmitted with the correct protection mode, as specified by the security policy [2, Section 3.6.2, pages 29–31].

To model these checks in CryptoVerif, we use a function get_policy such that get_policy($policy, msg$) is the protection mode (NONE, AUTH, or BOTH) of the message msg in the policy $policy$. The function get_policy is left unspecified, so that the result holds for any such function. In ProVerif, we cannot leave functions unspecified, we can only leave predicates unspecified (by declaring that they "**block**"), so we use a predicate policy_ok($policy, prot, msg$) to mean that $prot$ is the protection mode of msg in the policy $policy$.

These checks are essential for security. In particular, an unprotected message should *never* be accepted when the policy specifies a protection AUTH or BOTH, otherwise the authenticity of the message can be broken. These checks are correctly mentioned as mandatory in the body of the specification [2, Section 3.6.2, page 29]:

> When a secure session is established between communicating peer aircraft and ground entities, the receiving entity verifies all received messages to ensure correct application of the security policy.

However, they are omitted in the description of the protocol in [2, Attachment 7]. We recommend adding them there. Honeywell agrees that it may be helpful to include a reference to Section 3.6.2 in Attachment 7, but points out that the commentary in Attachment 7, Section 7.1 clearly states that the AMS protocol process steps are included as guidance and the provisions in the main body of the document must be followed precisely.

Moreover, since messages may be sent in the clear when a secure session cannot be established (Section II-B *8)*), the receiving entity has to accept messages independently of the security policy outside secure sessions. This behavior comes with security risks: if no secure session is established, for instance because the adversary jammed secure session initiation messages, then the adversary can send its messages to the receiving entity, which will accept them, so authenticity will be broken. As we already noted in Section II-B *8)*, security is guaranteed only when a secure session is established, and our models include only the behavior inside secure sessions. In this case, the receiving entity always verifies the correct application of the security policy.

10) MAC length. The specification is not explicit on what the aircraft does with the MAC length that is part of the *AlgSel* field of the AMS_Init_RSP message that it receives from the ground entity. If the protocol runs normally, the ground entity sends back the MAC length chosen by the aircraft. However, [2, Table 7.2.1-1] does not say what the aircraft should do when the MAC length it receives differs from the one it sent.

We stress that the aircraft should not set the MAC length based on *AlgSel* (in contrast to what it does for the MAC algorithm). Indeed, suppose that the aircraft sets its MAC length to the length received in *AlgSel*. The aircraft has initially sent a MAC length l in the field *Algos* of the AMS_Init_REQ message. If the adversary cannot forge MACs of length l but can forge MACs of another length

l', it can set the MAC length in *AlgSel* to l' and forge the MAC of the AMS_Init_RSP message for the length l' (either by modifying the AMS_Init_RSP message sent by the ground entity or by building its own). The aircraft would then accept this message. The adversary may then continue forging MACs of length l' for subsequent messages. This is a downgrading attack in which the achieved security level corresponds to the weakest MAC length (the shortest), even though the aircraft may have initially selected a stronger MAC length.

The following two behaviors avoid this problem:

- The aircraft ignores the MAC length in *AlgSel*, and always uses the MAC length it sent. Our models assume this behavior.
- The aircraft verifies that the MAC length in *AlgSel* is the same as the MAC length it sent. Otherwise, it discards the AMS_Init_RSP message.

Honeywell disagrees that the specification is not explicit on this point. They confirm that the MAC length is dictated by the aircraft and cannot be changed by the ground, and mention that the treatment of the MAC length is explained in [2, Table 7.2.1-1, Step N02, Item 1, Bullet 4]:

> $AlgID_U$, aircraft entity crypto-algorithm identifier, which indicates the *configured* MAC length, and the message authentication and encryption algorithms supported by the aircraft entity

and in [2, Section 5.2.2]:

> The AMS Algorithm Select Type (AMS_AlgSelType) is a 16-bit parameter type used to *confirm* the selected MAC length and [...]

We still recommend making explicit the reaction of the aircraft when the MAC length it receives differs from the one it sent, to avoid any misinterpretation of the standard.

C. Security Assumptions on Primitives

ProVerif uses the symbolic model, in which messages are terms, so the model cannot be accurate at the bit level. Bitstring concatenation is modeled as tuples. Cryptographic primitives are supposed to be perfect. Diffie-Hellman key agreements are modeled by the equation $d_U Q_V = d_V Q_U$; all other algebraic relations are ignored.

In contrast, CryptoVerif relies on the computational model of cryptography. The CryptoVerif model is exact at the bit level. It uses the following assumptions:

1) Certificates. The signatures used in certificates are unforgeable under chosen-message attacks (UF-CMA) [33]. These signatures may use a scheme different from the one used in the protocol to sign messages under d_U or d_V.

2) Signatures and Diffie-Hellman. The long-term key pairs of the aircraft and the ground entity are used for ECDSA signatures and for Diffie-Hellman key exchanges. Therefore, we need a joint security assumption on the signatures and the Diffie-Hellman key exchange, which is non-standard. We assume that ECDSA signatures are unforgeable under chosen-message attacks (UF-CMA) [24] and that the gap

Diffie-Hellman (GDH) assumption [43] holds with common keys.

For our anti-replay protection, we need strong unforgeability: if the signatures are just unforgeable, an adversary can replay the AMS_Init_REQ message by using a different signature for the same message, and thus pass our anti-replay protection. However, standard ECDSA signatures are malleable: if (r, s) is a valid signature, then so is $(r, (-s)$ $\mod n)$ where n is the curve order. This malleability can be avoided by requiring that s is less than $n/2$ [47]. For the protocol with replay protection, we assume this additional restriction, so that ECDSA signatures are strongly unforgeable under chosen-message attacks (SUF-CMA) [24] and that the gap Diffie-Hellman (GDH) assumption [43] holds with common keys. We do not make this stronger assumption for the original protocol, since the standard does not mention such a restriction. These assumptions are defined in CryptoVerif by allowing calls to signature oracles in the gap Diffie-Hellman assumption, and calls to Diffie-Hellman oracles in the (S)UF-CMA signature assumption (see the long version of this paper [22, Appendix A]).

Additionally, we need a collision resistance property for signatures: two signatures (generated with distinct fresh randomness) have a negligible probability of being equal. This property holds for ECDSA signatures, because if two signatures with randomness r_1 and r_2 respectively are equal, then $r_1.G$ and $r_2.G$ have the same first coordinate modulo n, where G is the considered group generator and n is its order. This property has a negligible probability of happening for independent random r_1 and r_2. This property is important for the security of the protocol: using deterministic signatures would remove any randomness from the AMS_Init_REQ message, so that two AMS_Init_REQ messages with the same timestamp would be equal, leading to the same replay attacks as those we find for the shared-key protocol (Section III-D).

3) Key derivation. We assume that the concatenation of the key derivation functions KDF256 and KDF128 defined by

$$\text{KDF256}(k, (X, U, V)) = \text{KDF}(k; 256;$$
$$\text{concatSI}(1, \text{HMAC-SHA256}, X, U, V, 256))$$
$$\text{KDF128}(k, (X, U, V)) = \text{KDF}(k; 128;$$
$$\text{concatSI}(2, \text{AES128-CFB128}, X, U, V, 128))$$

is a random oracle [11]. We justify this assumption in [22, Appendix B], assuming that the compression function of SHA256 is a random oracle. Using a single random oracle and splitting its result into two keys yields more efficient verification in CryptoVerif than using two independent random oracles.

4) MAC. We assume that the truncation of HMAC-SHA256 (to 32, 64, and 128 bits, as used in the protocol) is an SUF-CMA MAC. This property holds assuming that HMAC-SHA256 is a pseudo-random function (PRF). Indeed, the truncation of a PRF is also a PRF, and a PRF

is an SUF-CMA MAC [10, Section 2.4]. By [9], HMAC-SHA256 is a PRF if the compression function of SHA256 is a PRF, using its first argument as key, and also using its second argument as key under a related key attack.

5) Encryption. The initialization vector (IV) of the encryption scheme is computed as $IV = \mathsf{E}(KENC_{\mathsf{U},\mathsf{V}}, 0; IVdata)$ where $\mathsf{E}(k, IV, m)$ is the AES128-CFB128 encryption of m under the key k with IV IV and $IVdata$ is basically a counter [2, Table 7.3.1-1, Step D03, Item 1].

By definition of the CFB encryption mode [40], we have

$$IV = \mathsf{AES128}(KENC_{\mathsf{U},\mathsf{V}}, 0) \oplus IVdata \qquad (3)$$

where $\oplus$ is exclusive or. However, [40, Appendix C] says:

> For the CBC and CFB modes, the IVs must be unpredictable. In particular, for any given plaintext, it must not be possible to predict the IV that will be associated to the plaintext in advance of the generation of the IV.
> There are two recommended methods for generating unpredictable IVs. The first method is to apply the forward cipher function, under the same key that is used for the encryption of the plaintext, to a nonce. The nonce must be a data block that is unique to each execution of the encryption operation. [...] The second method is to generate a random data block using a FIPS-approved random number generator.

The generation of the IV in the AMS protocol does not follow these recommended methods for generating unpredictable IVs. The first recommended method would yield

$$IV = \mathsf{AES128}(KENC_{\mathsf{U},\mathsf{V}}, IVdata) \qquad (4)$$

Assuming that AES128 is a pseudo-random permutation (PRP) [38], formula (4) is indistinguishable from generating a fresh random IV for each new value of $IVdata$, so the IVs are unpredictable as desired.

In contrast, formula (3) allows one to predict the whole sequence of IVs knowing the first IV. Fortunately, the IVs remains secret in the AMS protocol, so we can still prove the protocol secure, at the cost of a less standard proof.

We assume that AES128 is a PRP. We let

$$\mathsf{ENC}(k, IVdata, m) = \mathsf{E}(k, \mathsf{E}(k, 0; IVdata); m)$$
$$\mathsf{DEC}(k, IVdata, c) = \mathsf{D}(k, \mathsf{E}(k, 0; IVdata); c)$$

and we show that $\mathsf{DEC}(k, IVdata, \mathsf{ENC}(k, IVdata, m)) = m$ and that the probability of distinguishing

$$\mathcal{O}_{\mathrm{gen}}() := k \xleftarrow{R} \mathsf{enc\_key}; \mathbf{return}(); \mathbf{foreach}\ i \le n\ \mathbf{do}$$
$$\mathcal{O}_{\mathrm{enc}}(IVdata, m) := \mathbf{return}(\mathsf{ENC}(k, IVdata, m)) \qquad (5)$$

from

$$\mathcal{O}_{\mathrm{gen}}() := k \xleftarrow{R} \mathsf{enc\_key}; \mathbf{return}(); \mathbf{foreach}\ i \le n\ \mathbf{do}$$
$$\quad \mathcal{O}_{\mathrm{enc}}(IVdata, m) :=$$
$$\quad \mathbf{find}\ j \le n\ \mathbf{suchthat\ defined}(IVdata[j]) \wedge$$
$$\quad\quad IVdata = IVdata[j]\ \mathbf{then\ event}\ \mathsf{repeatedIV}\ \mathbf{else}$$
$$\quad \mathbf{return}(\mathsf{ENC}(k, IVdata, Z(m)))$$
$$(6)$$

is at most the probability of event repeatedIV plus a negligible probability, where $Z(m)$ is the bitstring of the same length as m and consisting only of zeroes. The game (5) defines an oracle $\mathcal{O}_{\mathrm{gen}}$ that generates an encryption key k, chosen randomly with uniform probability in type enc_key. Then, this oracle returns control to the adversary and n copies of oracle $\mathcal{O}_{\mathrm{enc}}$ are defined, which return the encryption of message m using the IV data $IVdata$. The game (6) is similar but it executes event repeatedIV when $IVdata$ has been previously used, and encrypts $Z(m)$ instead of m. This property shows that, when the $IVdata$ is never reused, the adversary cannot distinguish the encryption of m from the encryption of $Z(m)$. Because CryptoVerif lacks loops, we prove this property only on an example with 3 messages ($n = 3$), containing respectively 1, 2, and 3 blocks. (Details on this proof can be found in [22, Appendix C].) It is easy to see manually that this result generalizes to any number of messages with any number of blocks. We use that general result as assumption in the proof of the protocol.

6) Hash function. We assume that SHA256 is collision-resistant. This property holds in particular when the compression function of SHA256 is collision-resistant [29].

D. Security Analysis

Having modeled the protocol, we can use ProVerif and CryptoVerif to verify its security properties:

1) Freshness of AMS_Init_REQ and AMS_Init_IND messages. ProVerif proves this freshness, by showing the following correspondences (7) and (8):

$$\mathbf{event}(\mathsf{recv\_Init\_REQ}(\mathsf{V}, \mathsf{U}, x, s_{\mathsf{U}}, t_{\mathsf{U}}, t_{\mathsf{V}})) \Longrightarrow$$
$$\mathsf{check\_time}(t_{\mathsf{U}}, t_{\mathsf{V}}) \wedge \qquad (7)$$
$$\mathbf{event}(\mathsf{send\_Init\_REQ}(\mathsf{U}, \mathsf{V}, x, s_{\mathsf{U}}, t_{\mathsf{U}})).$$

The event $\mathsf{recv\_Init\_REQ}(\mathsf{V}, \mathsf{U}, x, s_{\mathsf{U}}, t_{\mathsf{U}}, t_{\mathsf{V}})$ is executed when the ground entity V accepts an AMS_Init_REQ message x, s_{U} from the aircraft U at time t_{V}, and this message has timestamp t_{U}. (x is the message without signature, s_{U} is its signature.) Similarly, the event $\mathsf{send\_Init\_REQ}(\mathsf{U}, \mathsf{V}, x, s_{\mathsf{U}}, t_{\mathsf{U}})$ is executed when U sends an AMS_Init_REQ message x, s_{U} to V with timestamp t_{U}. Hence, this correspondence means that, if V accepts an AMS_Init_REQ message at time t_{V} from U, and this message has signature s_{U} and timestamp t_{U}, then the message has been sent by U to V, with the same signature s_{U}, at time t_{U} and $\mathsf{check\_time}(t_{\mathsf{U}}, t_{\mathsf{V}})$, that is, the inequality (2) holds.

The following correspondence proves a similar property for the AMS_Init_IND message:

$$\textbf{event}(\text{recv\_Init\_IND}(\mathsf{U},\mathsf{V},x,s_\mathsf{V},t_\mathsf{V},t_\mathsf{U})) \Longrightarrow$$
$$\text{check\_time}(t_\mathsf{V},t_\mathsf{U}) \wedge \qquad\qquad (8)$$
$$\textbf{event}(\text{send\_Init\_IND}(\mathsf{V},\mathsf{U},x,s_\mathsf{V},t_\mathsf{V})).$$

The authenticity of the timestamp comes from the signature, and the freshness is checked by the receiver of the message.

CryptoVerif proves slightly weaker properties, due to the malleability of ECDSA signatures. The signatures may be different on the ground and aircraft sides, so CryptoVerif proves the correspondences[1]:

$$\textbf{event}(\text{recv\_Init\_REQ}(\mathsf{V},\mathsf{U},x,s_\mathsf{U},t_\mathsf{U},t_\mathsf{V})) \Longrightarrow$$
$$\text{check\_time}(t_\mathsf{U},t_\mathsf{V}) \wedge \qquad\qquad (9)$$
$$\textbf{event}(\text{send\_Init\_REQ}(\mathsf{U},\mathsf{V},x,s_\mathsf{U}',t_\mathsf{U})).$$
$$\textbf{event}(\text{recv\_Init\_IND}(\mathsf{U},\mathsf{V},x,s_\mathsf{V},t_\mathsf{V},t_\mathsf{U})) \Longrightarrow$$
$$\text{check\_time}(t_\mathsf{V},t_\mathsf{U}) \wedge \qquad\qquad (10)$$
$$\textbf{event}(\text{send\_Init\_IND}(\mathsf{V},\mathsf{U},x,s_\mathsf{V}',t_\mathsf{V})).$$

For the protocol with replay protection, we assume SUF-CMA signatures, so CryptoVerif proves (7) and (8).

Obviously, for a proper functioning of the protocol, the clocks of the aircraft and of the ground entity must remain synchronized. An adversary could perform attacks if it managed to desynchronize these clocks; such attacks are outside the scope of our model. (Using nonces instead of timestamps could avoid such attacks.)

2) No replay for AMS_Init_REQ and AMS_Init_IND messages. To prove this property, we need to consider our improved replay protection, since the original protocol does not guarantee the absence of replays. This property is modeled by injective variants of (7) and (8):

$$\textbf{inj-event}(\text{recv\_Init\_REQ}(\mathsf{V},\mathsf{U},x,s_\mathsf{U},t_\mathsf{U},t_\mathsf{V})) \Longrightarrow$$
$$\text{check\_time}(t_\mathsf{U},t_\mathsf{V}) \wedge \qquad\qquad (11)$$
$$\textbf{inj-event}(\text{send\_Init\_REQ}(\mathsf{U},\mathsf{V},x,s_\mathsf{U},t_\mathsf{U})).$$
$$\textbf{inj-event}(\text{recv\_Init\_IND}(\mathsf{U},\mathsf{V},x,s_\mathsf{V},t_\mathsf{V},t_\mathsf{U})) \Longrightarrow$$
$$\text{check\_time}(t_\mathsf{V},t_\mathsf{U}) \wedge \qquad\qquad (12)$$
$$\textbf{inj-event}(\text{send\_Init\_IND}(\mathsf{V},\mathsf{U},x,s_\mathsf{V},t_\mathsf{V})).$$

These correspondences guarantee that each reception of the AMS_Init_REQ message on V corresponds to a *distinct* emission of that message on U, and similarly for AMS_Init_IND. Injectivity is enforced by the keyword **inj-event**.

In ProVerif, we prove these correspondences by combining (7) and (8) with a manual argument, which relies on the following observation.

Lemma 1: If the correspondence $\textbf{event}(e(x,M)) \Longrightarrow \phi \wedge \textbf{event}(e'(x,M'))$ holds and event $e(x,M)$ cannot be executed several times with the same value of x, then the injective correspondence $\textbf{inj-event}(e(x,M)) \Longrightarrow \phi \wedge \textbf{inj-event}(e'(x,M'))$ also holds.

[1]For uniformity, we use the ProVerif syntax for all correspondences. The actual syntax in CryptoVerif differs in minor ways.

Proof: If a trace contains n executions of $e(x,M)$, they are $e(x,M)\sigma_1, \ldots, e(x,M)\sigma_n$ for substitutions $\sigma_1, \ldots, \sigma_n$ with pairwise distinct values of $x\sigma_1, \ldots, x\sigma_n$. Then, by the correspondence $\textbf{event}(e(x,M)) \Longrightarrow \phi \wedge \textbf{event}(e'(x,M))$, for all $i \leq n$, we have that $\phi\sigma_i$ holds and $e'(x,M)\sigma_i$ is executed in this trace. Furthermore, these executions of $e'(x,M)\sigma_i$ are distinct for distinct i, because their arguments $x\sigma_i$ are distinct for distinct i. This property proves the injective correspondence $\textbf{inj-event}(e(x,M)) \Longrightarrow \phi \wedge \textbf{inj-event}(e'(x,M'))$. ∎

This observation is not new [19, before Corollary 3, page 406]. It obviously remains true when x and M are replaced with several variables and terms respectively, and when the arguments of events are reordered.

Our improved replay protection guarantees that the ground entity never accepts two AMS_Init_REQ messages with the same $\mathsf{U},s_\mathsf{U},t_\mathsf{U}$. Hence distinct events recv_Init_REQ(V,U, $x,s_\mathsf{U},t_\mathsf{U},t_\mathsf{V}$) have distinct triples $\mathsf{U},s_\mathsf{U},t_\mathsf{U}$, so by correspondence (7) and Lemma 1, (11) holds. The correspondence (12) is proved similarly. Indeed, ProVerif is unable to prove injectivity automatically when it is obtained using a replay cache: injectivity relies on the event recv_Init_REQ being executed only when a lookup in the replay cache fails. However, in its internal representation by Horn clauses, ProVerif overapproximates the behaviors and considers that this lookup may always fail, so the required information is lost.

In contrast, CryptoVerif treats lookup failures precisely, so in CryptoVerif, we model our improved replay check, which guarantees that the AMS_Init_REQ and AMS_Init_IND messages are never accepted with the same identity, timestamp, and signature. With this improved check, CryptoVerif shows (11) and (12).

3) Entity authentication. ProVerif and CryptoVerif prove the correspondence (7) or (9), which proves non-injective authentication of the aircraft to the ground entity. Furthermore, with our improved replay check, (11) proves injective authentication of the aircraft to the ground entity.

ProVerif and CryptoVerif also prove the correspondence:

$$\textbf{inj-event}(\text{recv\_Init\_RSP}(\mathsf{U},\mathsf{V},x,s_\mathsf{U},Rand_\mathsf{V},t_\mathsf{U},$$
$$KMAC_{\mathsf{U},\mathsf{V}}, KENC_{\mathsf{U},\mathsf{V}})) \Longrightarrow$$
$$\text{check\_time}(t_\mathsf{U},t_\mathsf{V}) \wedge \qquad\qquad (13)$$
$$\textbf{inj-event}(\text{send\_Init\_RSP}(\mathsf{V},\mathsf{U},x,s_\mathsf{U},Rand_\mathsf{V},$$
$$t_\mathsf{U},t_\mathsf{V},KMAC_{\mathsf{U},\mathsf{V}}, KENC_{\mathsf{U},\mathsf{V}})).$$

This correspondence means that, if U accepts an AMS_Init_RSP message x from V, then the message has been sent by V to U, and U and V share the same signature s_U, random number $Rand_\mathsf{V}$, time t_U, MAC key $KMAC_{\mathsf{U},\mathsf{V}}$, encryption key $KENC_{\mathsf{U},\mathsf{V}}$, and there is a time t_V at which V received the AMS_Init_REQ message such that check_time($t_\mathsf{U},t_\mathsf{V}$), that is, the inequality (2) holds. Furthermore, since the correspondence is injective (**inj-event**), each AMS_Init_RSP message received by U corresponds to a distinct AMS_Init_RSP message sent by V. So (13) proves authentication of the ground entity to the aircraft.

Hence, these correspondences prove mutual authentication of the aircraft and ground entity.

4) Message authentication. This property is formalized by the following correspondences:

$$\mathbf{inj\text{-}event}(\mathsf{recv\_msg}(x, y, t_\mathsf{U}, s_\mathsf{U}, Rand_\mathsf{V}, policy,$$
$$prot, enc, count, msg)) \Longrightarrow$$
$$prot = \mathsf{get\_policy}(policy, msg) \wedge$$
$$(\mathbf{inj\text{-}event}(\mathsf{send\_msg}(y, x, t_\mathsf{U}, s_\mathsf{U}, Rand_\mathsf{V},$$
$$policy, prot, enc, count, msg)) \vee prot = \mathsf{NONE})$$

$$(14)$$

where (x, y) is (U, V) or (V, U). These correspondences mean that, for (x, y) equal to (U, V) or (V, U), if x has received the payload message msg from entity y, protected under the protection mode $prot$, and the policy identifier is $policy$, then the protection mode associated to msg in the policy $policy$ is $prot$, and either $prot = \mathsf{NONE}$ or the message msg has been sent by y to x with the same timestamp t_U, signature s_U, random value $Rand_\mathsf{V}$, policy $policy$, protection mode $prot$, encryption algorithm enc, and counter $count$ on the receiver and sender sides. Furthermore, when $prot \neq \mathsf{NONE}$, the message has been sent at least as many times as it has been received. In other words, authenticity and absence of replays of the message are guaranteed when its associated policy is not NONE.

In ProVerif, since we use the predicate $\mathsf{policy\_ok}$ instead of the function $\mathsf{get\_policy}$ (see Section II-B *9)*), we prove the following correspondences:

$$\mathbf{event}(\mathsf{recv\_msg}(x, y, t_\mathsf{U}, s_\mathsf{U}, Rand_\mathsf{V}, policy,$$
$$prot, enc, count, msg)) \Longrightarrow$$
$$\mathsf{policy\_ok}(policy, prot, msg) \wedge$$
$$\mathsf{policy\_ok}(policy, prot', msg) \wedge$$
$$(\mathbf{event}(\mathsf{send\_msg}(y, x, t_\mathsf{U}, s_\mathsf{U}, Rand_\mathsf{V}, policy,$$
$$prot', enc, count, msg)) \vee prot = \mathsf{NONE})$$

where (x, y) is (U, V) or (V, U), and we infer (14) by combining them with the following manual argument. Since $\mathsf{policy\_ok}(policy, prot, msg)$ holds if and only if $prot = \mathsf{get\_policy}(policy, msg)$, we have $prot = prot' = \mathsf{get\_policy}(policy, msg)$, so

$$\mathbf{event}(\mathsf{recv\_msg}(x, y, t_\mathsf{U}, s_\mathsf{U}, Rand_\mathsf{V}, policy,$$
$$prot, enc, count, msg)) \Longrightarrow$$
$$prot = \mathsf{get\_policy}(policy, msg) \wedge$$
$$(\mathbf{event}(\mathsf{send\_msg}(y, x, t_\mathsf{U}, s_\mathsf{U}, Rand_\mathsf{V}, policy,$$
$$prot, enc, count, msg)) \vee prot = \mathsf{NONE}).$$

$$(15)$$

which is a non-injective variant of (14). We prove the absence of replays as follows. By the message count windowing criteria [2, Table 7.3.1-1, Step D08], the receiver accepts only once a message with a given count. Moreover, x, y identify the receiver and sender, respectively, and s_U and $Rand_\mathsf{V}$ identify uniquely the session of the aircraft and of the ground entity, respectively. (Since ECDSA

signatures are probabilistic, a new signature s_U is created at each session.) So distinct executions of $\mathsf{recv\_msg}(x, y, t_\mathsf{U}, s_\mathsf{U}, Rand_\mathsf{V}, policy, prot, enc, count, msg)$ (with $prot \neq \mathsf{NONE}$) have distinct values of the quintuple $x, y, s_\mathsf{U}, Rand_\mathsf{V}, count$. We have (15) and we prove injectivity as in Lemma 1, distinguishing the cases $prot \neq \mathsf{NONE}$ and $prot = \mathsf{NONE}$, so we obtain (14).

Since the receiver accepts a message with a given count only once, message counter roll-over must be avoided by ending the session and starting a new secure session when the counter reaches its maximal value of 255, as mentioned in [2, page 79].

CryptoVerif proves (14) directly.

5) Secrecy of keys. We study the secrecy of the keys $KMAC_{\mathsf{U},\mathsf{V}}$ and $KENC_{\mathsf{U},\mathsf{V}}$. In the symbolic model, ProVerif shows that the adversary cannot compute these keys (by showing that a constant message encrypted under these keys remains secret). In the computational model, we would like to show that these keys are indistinguishable from fresh independent random keys at the end of the session initiation. CryptoVerif proves this property for keys $KENC_{\mathsf{U},\mathsf{V}}$. This property does not hold for $KMAC_{\mathsf{U},\mathsf{V}}$, because this key is used in the second message of the session initiation. This point does not correspond to an attack against the protocol, but it makes it more difficult to prove security modularly: we cannot show that the session initiation provides a secure key, and then use this property in the application data exchange.

6) Secrecy of messages. To prove secrecy in ProVerif, we modify our model slightly so that the payload message is not initially known to the adversary. We define a private function (a function that cannot be applied by the adversary) msg that we use to build the payload message. The term $\mathsf{msg}(\mathsf{U}, \mathsf{V}, policy, x)$ represents a message sent from U to V under the policy $policy$. When encryption is supported, ProVerif proves the property:

$$attacker(\mathsf{msg}(\mathsf{U}, \mathsf{V}, policy, x)) \Longrightarrow$$
$$\mathsf{policy\_ok}(policy, \mathsf{AUTH}, \mathsf{msg}(\mathsf{U}, \mathsf{V}, policy, x)) \vee \quad (16)$$
$$\mathsf{policy\_ok}(policy, \mathsf{NONE}, \mathsf{msg}(\mathsf{U}, \mathsf{V}, policy, x)).$$

and the symmetric correspondence obtained by swapping U and V. In other words, if the attacker has the payload message sent from U to V or from V to U, then the policy specifies a protection AUTH or NONE for this message. So, if the policy specifies a protection BOTH, then the attacker cannot have the payload message, so this message is secret.

Obviously, when encryption is not supported, the messages cannot be secret. ProVerif confirms this point.

In CryptoVerif, we suppose that encryption is supported. CryptoVerif then proves the following secrecy property. At the beginning of the protocol, we choose a random bit b with uniform probability. For each payload message, the sender actually gets two candidate payloads $payload_0$ and $payload_1$ from the adversary. When the current session is between the honest entities U and V and the two payloads have protection mode BOTH (that is, $\mathsf{get\_policy}(policy, payload_0) = \mathsf{get\_policy}(policy, payload_1) = \mathsf{BOTH}$), we require that

the corresponding encoded and compressed payloads have the same length. Otherwise, we require that $payload_0 = payload_1$. The sender then sends $payload_b$ following the protocol. We show that the random bit b is secret, that is, the adversary has a probability close to 1/2 of guessing b. So the adversary only has a negligible probability of distinguishing whether $payload_0$ or $payload_1$ has been sent, which proves secrecy for messages with associated protection mode BOTH in sessions between U and V.

Although the content of the messages is secret, the length of the encoded and compressed payload is public, since encryption preserves the length. In some cases, this length may reveal important information about the underlying message. (See [44] for an attack that relies on the length of encrypted messages in a different context.) From the point of view of security, it would be stronger to hide the length of the messages as well, for instance by arranging so that all messages whose length is within a certain interval encrypt into a ciphertext of the same length, in the style of the countermeasure proposed in [44]. However, a loss of bandwidth is unavoidable in this case.

7) Unknown key share attacks. ProVerif and CryptoVerif prove the following correspondence:

$$
\begin{aligned}
&\textbf{event}(\mathsf{send\_Init\_RSP}(V, U, x, s_\mathsf{U}, Rand_\mathsf{V}, t_\mathsf{U}, t_\mathsf{V}, \\
&\quad KMAC_\mathsf{U,V}, KENC_\mathsf{U,V})) \land \\
&\textbf{event}(\mathsf{recv\_Init\_RSP}(U', V', x', s'_\mathsf{U}, Rand'_\mathsf{V}, t'_\mathsf{U}, \\
&\quad KMAC'_\mathsf{U,V}, KENC_\mathsf{U,V})) \Longrightarrow U = U' \land V = V'.
\end{aligned} \quad (17)
$$

(An encoding is needed in ProVerif because it does not support conjunctions of events on the left-hand side of the arrow, but the proved correspondence is equivalent to the one above.) This correspondence shows that, if executions of U and V share the same encryption key, then U believes he talks to V and conversely. A similar correspondence holds for executions that share the same MAC key. These correspondences show the absence of unknown key share (UKS) attacks [27], [30]. In these attacks, the aircraft U believes it shares a key with a (possibly dishonest) ground entity V' and the ground entity V believes it shares a key with a (possibly dishonest) aircraft U' but the two sessions share the same keys, so the adversary can make a message from U to V' appear to V as a message from U', and similarly swapping U and V. (In bilateral UKS attacks [27], $U \neq U'$ and $V \neq V'$; in unilateral UKS attacks [30], $U \neq U'$ and $V = V'$ or symmetrically.)

8) Forward secrecy of messages. Forward secrecy means that, even if a long-term secret key is compromised after the protocol is executed, the exchanged messages remain secret. ProVerif finds an attack against this property. Indeed, if an adversary manages to compromise a secret key d_U or d_V even after the session has been terminated, it will be able to compute the shared keys and decrypt the messages, because the entities always reuse the same secret keys d_U and d_V, without including ephemerals in the Diffie-Hellman key agreement. We did not try to prove forward secrecy in

CryptoVerif, since we know that it does not hold from the symbolic analysis.

9) Key compromise impersonation. When the long-term key d_U of an aircraft is compromised, we naturally expect that the adversary can impersonate the aircraft to any ground entity. However, ProVerif finds out that the adversary can also impersonate any ground entity to the compromised aircraft: it finds an attack against the correspondence (13). This is a key compromise impersonation (KCI) attack. To perform this attack, the adversary runs the protocol exactly as a ground entity, except that it computes $Z_\mathsf{U,V}$ as $d_\mathsf{U}Q_\mathsf{V}$ using the compromised secret key d_U of the aircraft and the public key Q_V of the ground entity, instead of the normal computation of $Z_\mathsf{U,V} = d_\mathsf{V}Q_\mathsf{U}$. Since $d_\mathsf{U}Q_\mathsf{V} = d_\mathsf{V}Q_\mathsf{U}$, the two computations yield the same result. Furthermore, the computation of $Z_\mathsf{U,V}$ is the only point at which the secret key d_V of the ground entity is normally used, so the adversary succeeds in impersonating the ground entity.

When the long-term key d_V of a ground entity is compromised, ProVerif shows that the correspondence (7) still holds. Indeed, the protocol requires a signature of the aircraft, so the adversary cannot fully impersonate an aircraft to the ground entity, because it cannot forge a signature. However, ProVerif also finds out that the correspondence (15) does not hold when (x, y) is (U, V): the ground entity may receive messages that seem to come from the aircraft even though the aircraft never sent them. Indeed, after the first two messages of the protocol, the adversary can compute $Z_\mathsf{U,V} = d_\mathsf{V}Q_\mathsf{U}$, and from that compute the encryption and MAC keys, so it can decrypt and forge messages.

We did not try to prove the absence of key compromise impersonation attacks in CryptoVerif, since we know that it does not hold from the symbolic analysis.

10) Strengthened protocol. Moreover, attacks against these properties are not very serious as they only appear after the compromise of a long-term key, which is already a serious situation. However, they make this compromise even more serious.

These attacks can be avoided by using a signed Diffie-Hellman key exchange in which the long-term keys are used only for signing and Diffie-Hellman exponents are fresh in each session. Hence, we define a fixed protocol as follows:

Aircraft U		*Ground entity* V

Secure session initiation

<div align="center">

AMS_Init_REQ message
$m_1 = (H_1, \mathsf{Init\_REQ}, Policy, U, V, R_\mathsf{U}, Algos, t_\mathsf{U})$
$s_\mathsf{U} = \mathsf{SIG}(d_\mathsf{U}, (U, V, m_1))$

—————————————————————————————→

AMS_Init_RSP message
$m_2 = (H_2, \mathsf{Init\_RSP+}, AlgSel, R_\mathsf{V}, [cert_\mathsf{V}], t_\mathsf{U}, Rand_\mathsf{V})$
$\mathsf{SIG}(d_\mathsf{V}, (V, U, m_2, m_1, s_\mathsf{U}))$

←—————————————————————————————

</div>

where $e_\mathsf{U}, R_\mathsf{U}$ is a fresh ephemeral private-public key pair for the aircraft and $e_\mathsf{V}, R_\mathsf{V}$ is a fresh ephemeral private-public

key pair for the ground entity, and we compute

$$Z_{\mathsf{U},\mathsf{V}} = e_{\mathsf{V}} R_{\mathsf{U}} = e_{\mathsf{U}} R_{\mathsf{V}}$$
$$X_{\mathsf{U},\mathsf{V}} = \mathsf{HASH}((AlgSel, Algos, t_{\mathsf{U}}, s_{\mathsf{U}}, Rand_{\mathsf{V}}))\,.$$

The inclusion of m_1 in the signature of the AMS_Init_RSP message guarantees that U and V have common parameters (in particular the ephemeral R_{U} and the algorithms $Algos$) even when U's long-term key is compromised. The addition of $AlgSel$ in the computation of $X_{\mathsf{U},\mathsf{V}}$ guarantees that U and V share the same algorithms even when V's long-term key is compromised. We could not fully verify the fixed protocol in ProVerif because it took more than 8 Gb of memory. Indeed, the protocols verified in this paper are fairly intensive for ProVerif, because there are many cases. The fixed public key protocol is even more, probably because of the presence of additional randomness (the ephemerals), which leads to bigger terms. However, we proved using CryptoVerif that it satisfies all properties we consider in the computational model, which is stronger than a symbolic proof in ProVerif. The assumptions are the same as for the original protocol, except that we do not need a joint security assumption between Diffie-Hellman key agreements and signatures: we use SUF-CMA signatures and the computational Diffie-Hellman assumption (for most properties) or the stronger gap Diffie-Hellman assumption (for resistance to KCI attacks when the ground entity is compromised and for forward secrecy; these properties do not hold under the CDH assumption). Obviously, the stronger security properties come at a price: the key exchange of the fixed protocol is more costly than the original one, since U and V have to choose an ephemeral secret key and compute and send the corresponding ephemeral public key, and a MAC is replaced with a signature.

Another possible way to strengthen the protocol would be to use a triple Diffie-Hellman exchange, with ephemerals and without signatures, as in Signal [32], for instance. We preferred using a signed Diffie-Hellman exchange to be closer to the original protocol: the signatures authenticate the AMS_Init_IND and AMS_Init_REQ messages, as in the original protocol; that would not be true with a triple Diffie-Hellman exchange.

Honeywell made several comments on our proposal:

- The AMS security protocol is based on the technical provisions and security protocol specified for the Aeronautical Telecommunications Network (ATN). Although the ATN does not currently include provisions for confidentiality (encryption), the AMS encryption implementation is consistent with a white paper [26] that was presented to ICAO to demonstrate extensibility of the ATN security solution to support encryption in the future. The approach presented in [26] does not consider use of ephemerals.
- The current AMS security protocol is optimized to take into account the bandwidth, computational, and memory constraints of avionic systems.
- Our proposed use of ephemerals would require further assessment in the following areas: the ability of legacy avionics to generate quality ephemeral key pairs; the impact of increased overhead to exchange ephemeral public components during secure session initiation; and the impact on avionics computing performance and memory as a result of more complex computation of the secret key agreement value ($Z_{\mathsf{U},\mathsf{V}}$).
- With the current AMS security solution, session-specific encryption and authentication keys are never stored but may be recreated using archived copies of the session initiation messages plus the static public/private key pair of the ground entity. This may be necessary to decrypt archived encrypted messages in the event of an accident investigation. The addition of ephemeral keys would require the ground entity to also archive its public/private ephemerals (consistent with message retention policy) associated with each secure session.

11) Proof indications. In CryptoVerif, the proofs of the AMS public-key protocol require the user to give the main proof steps: apply the assumption on the signatures in certificates, use the random oracle assumption on the concatenation of KDF256 and KDF128, remove assignments on variable $Z_{\mathsf{U},\mathsf{V}}$ in the aircraft process and in the ground entity process to make equality tests such as $m = d_{\mathsf{V}} Q_{\mathsf{U}}$ appear, use the gap Diffie-Hellman assumption to prove that some of these tests are false, apply the assumption on signatures under d_{V} and d_{U}, and finally use the automatic proof strategy to conclude the proof. The proofs for the strengthened protocol use a similar style of guidance, sometimes with additional case distinctions. We refer the interested reader to the CryptoVerif scripts for details.

12) Runtime. In total, the analysis of the public-key protocol took 10 h 47 min in ProVerif and 17 min in CryptoVerif. The analysis of the strengthened protocol exhausted memory in ProVerif and took 1 h 16 min in CryptoVerif on an Intel Xeon 1.2 GHz. In this example, CryptoVerif is more efficient than ProVerif, probably because ProVerif explores an abstraction of all states of the protocol, while CryptoVerif performs a proof by game transformation, so it does not need to explore all these states. The guidance also helps CryptoVerif to be more efficient by avoiding the exploration of unsuccessful proofs.

III. AMS PROTOCOL USING A SHARED KEY

In this section, we study the AMS protocol using a shared key as defined in [2, Attachment 8].

A. Protocol Description

In this protocol, the aircraft and the ground entity initially share a long-term key $K_{\mathsf{U},\mathsf{V}}$. They establish session keys as shown in Figure 2. The protocol is similar to the public-key protocol (see Section II-A), but the AMS_Init_IND and AMS_Init_REQ messages are protected by MACs under $K_{\mathsf{U},\mathsf{V}}$ instead of signatures. The session keys are derived as follows:

$$X_{\mathsf{U},\mathsf{V}} = \mathsf{HASH}((Algos, t_{\mathsf{U}}, mac_{0,\mathsf{U}}, Rand_{\mathsf{V}}))$$

and $KMAC_{\mathsf{U},\mathsf{V}}$ and $KENC_{\mathsf{U},\mathsf{V}}$ are computed as in the public-key protocol except that $Z_{\mathsf{U},\mathsf{V}}$ is replaced with $K_{\mathsf{U},\mathsf{V}}$.

Aircraft U   _Ground entity_ V

Ground-initiated trigger

AMS_Init_IND message
$m_0 = (H_0, \mathsf{Init\_IND}, \mathsf{U}, \mathsf{V}, t_\mathsf{V})$
$mac_{0,\mathsf{V}} = \mathsf{MAC}(K_{\mathsf{U},\mathsf{V}}, (\mathsf{V}, \mathsf{U}, m_0))$
⟵

Secure session initiation

AMS_Init_REQ message
$m_1 = (H_1, \mathsf{Init\_REQ}, Policy, \mathsf{U}, \mathsf{V}, Algos, t_\mathsf{U})$
$mac_{0,\mathsf{U}} = \mathsf{MAC}(K_{\mathsf{U},\mathsf{V}}, (\mathsf{U}, \mathsf{V}, m_1))$
⟶

AMS_Init_RSP message
$m_2 = (H_2, \mathsf{Init\_RSP}+, AlgSel, t_\mathsf{U}, Rand_\mathsf{V})$
$\mathsf{MAC}(KMAC_{\mathsf{U},\mathsf{V}}, (\mathsf{V}, \mathsf{U}, m_2, Algos, t_\mathsf{U}, mac_{0,\mathsf{U}}))$
⟵
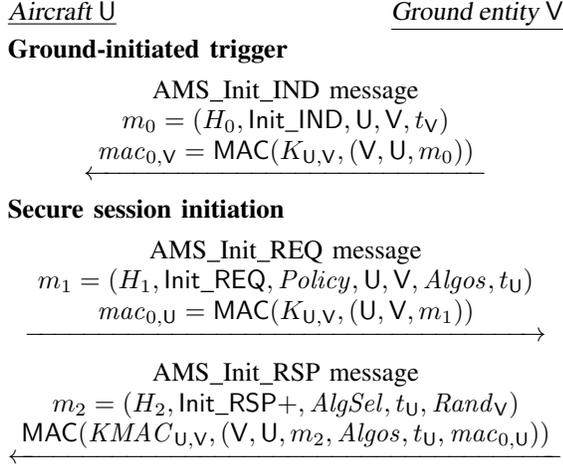
Figure 2.   Simplified AMS protocol using a shared key

Protocols	O	ST	RP
Freshness of AMS_Init_REQ (7)	Y	Y	Y
Freshness of AMS_Init_IND (8)	Y	Y	Y
No replay for AMS_Init_REQ (11)			Y
No replay for AMS_Init_IND (12)			Y
Non-injective authentication of V to U	Y	Y	Y
Injective authentication of V to U (13)		Y	Y
Non-injective message authentication (15)	Y	Y	Y
Injective message authentication (14)		Y	Y
Secrecy of $KMAC_{\mathsf{U},\mathsf{V}}$	PV	PV	PV
Secrecy of $KENC_{\mathsf{U},\mathsf{V}}$	PV	Y	Y
Secrecy of messages	PV	Y	Y
No unknown key share attacks (17)	Y	Y	Y

Figure 3.   Analysis of the shared-key protocol

The application data exchange proceeds exactly as in the public-key protocol.

B. Modeling Notes

Our comments of Section II-B also apply to the shared-key protocol, with the following exceptions:

1) Sessions. Except for unknown key share attacks, the models consider a single aircraft U and a single ground entity V that share a key $K_{\mathsf{U},\mathsf{V}}$ and talk to each other. All other executions of the protocol are considered as part of the adversary, since they use a different shared key. For unknown key share attacks, U and V may also talk to dishonest participants, as for the public-key protocol.

2) Certificates. Section II-B *3)* does not apply, since this protocol does not use certificates.

3) Replay protection. The shared-key protocol uses a replay check similar to the one used by the public-key protocol, and it suffers from the same weakness. However, the correction is slightly different, because in the shared-key protocol, the AMS_Init_IND and AMS_Init_REQ messages do not contain anything random. (In the public-key protocol, they contain a randomized signature.) Since the timestamp has a 2-second resolution, AMS_Init_IND or AMS_Init_REQ messages sent within the 2-second interval in which the timestamp does not change are equal. Hence, the receiver cannot distinguish a replay from a legitimate attempt at starting a session again within that interval.

To solve this problem, we assume that the aircraft never sends two AMS_Init_REQ messages to the same ground entity V with the same timestamp t_U. (Using a random nonce in addition to the timestamp t_U would guarantee that two sessions never start with the same AMS_Init_REQ message. However, it is an incompatible change in the protocol.)

Using this assumption, for each aircraft U, the ground entity should remember the timestamp t_U of the last AMS_Init_REQ message received from U, as long as $t - 120$ seconds $\leq t_\mathsf{U}$ where t is the ground entity's local time. If the timestamp t_U of a new AMS_Init_REQ

message received from U is not strictly more recent than the remembered timestamp t_U, the message should be discarded. (Replays of AMS_Init_REQ messages older than 120 seconds are already discarded by checking the timestamp.)

Moreover, as for the public-key protocol, the original protocol has no protection against replays of a ground-initiated trigger (AMS_Init_IND message). To avoid replay attacks, we recommend protecting against replays of the AMS_Init_IND message by making sure that the ground entity never sends two AMS_Init_IND messages with the same timestamp t_V and using a check similar to the one mentioned above for the AMS_Init_REQ message.

C. Security Assumptions on Primitives

The shared secret key $K_{\mathsf{U},\mathsf{V}}$ is used as key both for the MAC and for the key derivation function KDF. Using the same key for different cryptographic schemes is problematic for proving the protocol in the computational model: one cannot prove that the protocol is secure under standard security assumptions, namely that the MAC is SUF-CMA and the key derivation function is a pseudo-random function (PRF). In this case, one needs a combined security assumption about the two schemes. (A correction to this problem would be to use the key derivation function KDF also to derive the key of the MAC in the AMS_Init_IND and AMS_Init_REQ messages. However, this correction would alter the protocol in an incompatible way.)

To study the protocol in CryptoVerif, we prove that HMAC-SHA256 and the two key derivation functions KDF256 and KDF128 are indistinguishable from independent pseudo-random functions, even when they use the same key k, in the following two situations:

- when the compression function of SHA256 is a random oracle;
- when the compression function of SHA256 is a pseudo-random function and, given a 256-bit random key k, $\mathsf{SHA256\_comp}(\mathsf{SHA256\_init}, k \oplus \mathsf{opad})$ and $\mathsf{SHA256\_comp}(\mathsf{SHA256\_init}, k \oplus \mathsf{ipad})$ are independent pseudo-random bitstrings, and the function $x \mapsto \mathsf{SHA256\_comp}(\mathsf{SHA256\_init}, k\|x)$ is a pseudo-random function independent from those two bitstrings.

(SHA256_comp is the compression function of SHA256, SHA256_init is the initial value for SHA256 given in [41, Section 5.3.2], $\oplus$ is exclusive or, opad and ipad are distinct fixed 512-bits constants consisting of 64 copies of the bytes 0x5C and 0x36 respectively [42], x is 256-bit long, $\|$ is bitstring concatenation.)

The first assumption is very strong, but standard. The second assumption is not so strong but less standard. It still seems reasonable. The second result generalizes the proof that HMAC is a PRF given in [9]. These two results are proved using CryptoVerif [22, Appendix D].

Knowing that HMAC-SHA256, KDF256, and KDF128 are independent PRFs (even when they use the same key), we show that

- HMAC-SHA256 is SUF-CMA, and KDF256 and KDF128 are independent PRFs (even when they use the same key). This result generalizes the well-known result that a PRF is an SUF-CMA MAC [10, Section 2.4]. It is proved using CryptoVerif [22, Appendix E].
- HMAC-SHA256 has the following collision-resistance property: when the key k is used only for MAC queries, $\mathsf{MAC}(k, m_1) = \mathsf{MAC}(k, m_2)$ implies $m_1 = m_2$ with overwhelming probability, where MAC is HMAC-SHA256. This result is proved using CryptoVerif [22, Appendix F].
- The truncation of HMAC-SHA256 (to 32, 64, and 128 bits, as used in the protocol) is also an SUF-CMA MAC. This result holds because the truncation of a PRF is also a PRF, and a PRF is an SUF-CMA MAC.

We use these properties in the proof of protocol. We also assume that SHA256 is collision-resistant and we make the same assumption on the AES128-CFB128 encryption scheme as in the public-key protocol.

D. Security Analysis

Much like for the public-key protocol, we use ProVerif and CryptoVerif to verify the security properties of the protocol. For this protocol, CryptoVerif concludes without user guidance. Our results are summarized in Figure 3. We refer to correspondences used for the public-key protocol in this figure; still, the correspondences used for the shared-key protocol differ in that the arguments s_U and s_V are omitted from events and in (7), (8), (11), and (12), the argument x contains the AMS_Init_IND or AMS_Init_REQ message including the MAC.

The authentication of U to V does not appear in the figure as it corresponds to the freshness and absence of replays for AMS_Init_REQ. The non-injective authentication of V to U is modeled by a non-injective variant of (13). A shared-key protocol cannot provide forward-secrecy nor protect against key compromise impersonation: the compromise of $K_{\mathsf{U},\mathsf{V}}$ allows the adversary to compute all keys, decrypt all messages, and impersonate participants from both sides. So we do not test these properties.

In Figure 3, we consider three variants of the protocol: the original protocol (O), the protocol with the additional

assumption that the aircraft never sends two AMS_Init_REQ messages with the same V and t_U (ST, for "Single Timestamp"), and the protocol with our improved replay protection (RP). We write Y (Yes) when the property is proved both by ProVerif and CryptoVerif (using manual arguments for proving injectivity in ProVerif as for the public-key protocol), and PV when it is proved only by ProVerif.

To obtain absence of unknown key share attacks in CryptoVerif, we had to assume that KDF256 and KDF128 are independent random oracles, as for the public-key protocol. In the computational model, $KMAC_{\mathsf{U},\mathsf{V}}$ is not secret, that is, it is distinguishable from random, because it is used in the AMS_Init_RSP message, as in the public-key protocol.

All other properties that do not hold are due to replay attacks. As discussed in Section III-B *3)*, absence of replays for AMS_Init_REQ and AMS_Init_IND messages holds only with our improved protection against replays. Furthermore, consider the following scenario shown in Figure 4. The aircraft U and the ground entity V first run a secure session. In the 2-second interval during which the timestamp remains the same, the aircraft starts a new secure session with V with the same t_U, so exactly the same AMS_Init_REQ message. (This scenario is not entirely unrealistic since, in case the first session has been triggered by an AMS_Init_IND message from the ground entity, the adversary can force such a secure session restart by replaying the AMS_Init_IND message as mentioned in Section II-B *6)*.) The adversary intercepts this AMS_Init_REQ message and replies by replaying the AMS_Init_RSP message of the first session. This message is accepted by the aircraft, so that the aircraft executes event recv_Init_RSP twice, while the ground entity executes event send_Init_RSP once. This point breaks injective authentication of V to U. The established sessions keys are the same in both sessions, which breaks secrecy of $KENC_{\mathsf{U},\mathsf{V}}$ in the computational model: indeed, the keys $KENC_{\mathsf{U},\mathsf{V}}$ are each indistinguishable from random, but they are not independent of each other, which we require for computational secrecy. Furthermore, since the keys are equal, application data messages of the first session can be replayed in the second one, which breaks injective message authentication. Finally, if the aircraft sends encrypted data in this second session, the adversary can break secrecy of messages in the computational model, as follows.

In both sessions, for the n-th message sent by the aircraft, the IV data is $IVdata(n) = (0x00\|n\|\text{zero-pad-bits})$, so the IV is also the same in both sessions:

$$IV(n) = \mathsf{E}(KENC_{\mathsf{U},\mathsf{V}}, 0; IVdata(n)).$$

Suppose that the k-th block of plaintext (that is, encoded and compressed payload) of the n-th message sent by the aircraft in the first session is P_k, and in the second session P'_k. Similarly, let C_k be the k-th block of ciphertext of the n-th message sent by the aircraft in the first session, and C'_k in the second session. By definition of AES128-CFB128,

$$C_1 = P_1 \oplus \mathsf{AES128}(KENC_{\mathsf{U},\mathsf{V}}, IV(n)) \text{ and}$$
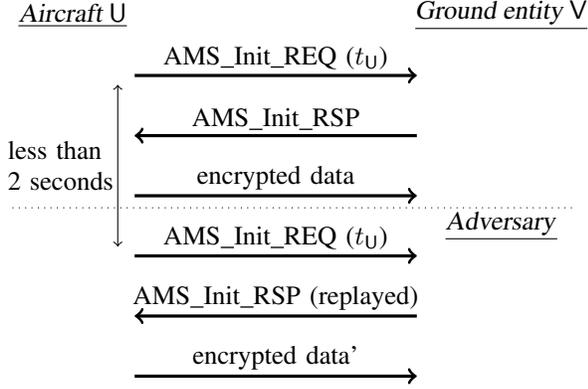$$C'_1 = P'_1 \oplus \mathsf{AES128}(KENC_{\mathsf{U},\mathsf{V}}, IV(n))$$

Figure 4. Replay attack

so $C_1 \oplus C_1' = P_1 \oplus P_1'$. The adversary knows C_1 and C_1', so it can compute $C_1 \oplus C_1'$ and thus obtain $P_1 \oplus P_1'$. This is already a breach of secrecy, as the adversary obtains the exclusive or of two blocks of plaintext. In case $C_1 \neq C_1'$, that is, $P_1 \oplus P_1' \neq 0$, the attack stops here because

$$C_2 = P_2 \oplus \mathsf{AES128}(KENC_{\mathsf{U},\mathsf{V}}, C_1)$$
$$C_2' = P_2' \oplus \mathsf{AES128}(KENC_{\mathsf{U},\mathsf{V}}, C_1')$$

so P_2 and P_2' are xor-ed with different encryptions. However, in case $C_1 = C_1'$, that is, $P_1 \oplus P_1' = 0$, the attack continues until a block differs: if for all $j < k$, $P_j = P_j'$, then for all $j < k$, $C_j = C_j'$ by an easy induction on j since $C_j = P_j \oplus \mathsf{AES128}(KENC_{\mathsf{U},\mathsf{V}}, C_{j-1}) = P_j' \oplus \mathsf{AES128}(KENC_{\mathsf{U},\mathsf{V}}, C_{j-1}') = C_j'$, and

$$C_k = P_k \oplus \mathsf{AES128}(KENC_{\mathsf{U},\mathsf{V}}, C_{k-1})$$
$$C_k' = P_k' \oplus \mathsf{AES128}(KENC_{\mathsf{U},\mathsf{V}}, C_{k-1}')$$
$$= P_k' \oplus \mathsf{AES128}(KENC_{\mathsf{U},\mathsf{V}}, C_{k-1})$$

so $C_k \oplus C_k' = P_k \oplus P_k'$, which can be computed by the adversary. Hence, the adversary can test whether the n-th encrypted message sent by the aircraft in the second session is the same as the corresponding message of the first session, and in case these messages differ, it gets the exclusive or of the first blocks of plaintext that differ. This attack does not appear in the symbolic model, both because the symbolic model ignores the details of the encryption algorithm and because we verified a weaker secrecy property: we showed that the adversary cannot recover the plaintext.

Therefore, replays may have more serious consequences in the shared-key protocol than in the public-key protocol. The analysis of the shared-key protocol took 8 min in ProVerif and 11 min in CryptoVerif on an Intel Xeon 1.2 GHz.

IV. CONCLUSION

Our study confirms the main security properties of the public-key protocol: authentication and secrecy of the messages (provided messages are not sent in the clear due to the failure of secure session initiation). It leads to several recommendations for changes in the ARINC823 standard: fix the timestamp verification formula (1); make explicit the treatment of the MAC length received by the aircraft; make explicit the policy check for received messages; discuss the security risks associated to sending messages in the clear when the secure session initiation fails, and their possible mitigations; strengthen the replay protection. The last point is especially important for the shared-key protocol, in which replays may lead to secrecy breaches and replays of application data messages. We designed a strengthened public-key protocol, which satisfies forward secrecy, protects against key compromise impersonation attacks, and avoids sharing keys between several cryptographic primitives.

This case study also illustrates the complementarity between symbolic and computational verification. ProVerif allows us to easily find most attacks, but the attack against secrecy in the shared-key protocol does not appear at the symbolic level, and was discovered due to the failure of a proof in CryptoVerif. The computational analysis provides stronger security guarantees, but is also more difficult. The high level of automation of CryptoVerif was very helpful for that analysis, even if a bit of guidance was needed for the proofs of the public-key protocol. CryptoVerif came to the rescue in cases that exceeded ProVerif's capabilities (dealing with some equations; analyzing our strengthened protocol).

REFERENCES

[1] M. Aizatulin, A. D. Gordon, and J. Jürjens, "Computational verification of C protocol implementations by symbolic execution," in *19th ACM Conference on Computer and Communications Security (CCS'12).* ACM Press, Oct. 2012, pp. 712–723.

[2] "ARINC specification 823P1: Datalink security, part 1 ACARS message security," Dec. 2007, available for sale at http://aviation-ia.com/cf/store/catalog_detail.cfm?item_id=872.

[3] A. Armando *et al.*, "The AVISPA tool for automated validation of Internet security protocols and applications," in *Computer Aided Verification, 17th International Conference, CAV 2005*, ser. LNCS, K. Etessami and S. K. Rajamani, Eds., vol. 3576. Springer, Jul. 2005, pp. 281–285.

[4] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P.-Y. Strub, "EasyCrypt: A tutorial," in *Foundations of Security Analysis and Design VII*, ser. LNCS, A. Aldini, J. Lopez, and F. Martinelli, Eds. Springer, 2014, vol. 8604, pp. 146–166.

[5] G. Barthe, C. Fournet, B. Grégoire, P. Strub, N. Swamy, and S. Z. Béguelin, "Probabilistic relational verification for cryptographic implementations," in *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* ACM, 2014, pp. 193–206.

[6] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin, "Computer-aided security proofs for the working cryptographer," in *Advances in Cryptology – CRYPTO 2011*, ser. LNCS, P. Rogaway, Ed., vol. 6841. Springer, Aug. 2011, pp. 71–90.

[7] G. Barthe, B. Grégoire, and S. Zanella, "Formal certification of code-based cryptographic proofs," in *36th ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL'09).* ACM, Jan. 2009, pp. 90–101.

[8] S. Z. Béguelin, B. Grégoire, G. Barthe, and F. Olmedo, "Formally certifying the security of digital signature schemes," in *30th IEEE Symposium on Security and Privacy, S&P 2009*. IEEE, May 2009, pp. 237–250.

[9] M. Bellare, "New proofs for NMAC and HMAC: Security without collision-resistance," in *CRYPTO'06*, ser. LNCS, C. Dwork, Ed., vol. 4117. Springer, Aug. 2006, pp. 602–619, extended version available at http://cseweb.ucsd.edu/~mihir/papers/hmac-new.pdf.

[10] M. Bellare, J. Kilian, and P. Rogaway, "The security of the cipher block chaining message authentication code," *Journal of Computer and System Sciences*, vol. 61, no. 3, pp. 362–399, Dec. 2000.

[11] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," in *Computer and Communications Security (CCS'93)*. ACM Press, 1993, pp. 62–73.

[12] J. Bengtson, K. Bhargavan, C. Fournet, A. Gordon, and S. Maffeis, "Refinement types for secure implementations," *ACM Transactions on Programming Languages and Systems*, vol. 33, no. 2, 2011.

[13] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the tls 1.3 standard candidate," in *IEEE Symposium on Security and Privacy (S&P'17)*. IEEE, May 2017, to appear.

[14] K. Bhargavan, R. Corin, C. Fournet, and E. Zălinescu, "Cryptographically verified implementations for TLS," in *15th ACM Conference on Computer and Communications Security (CCS'08)*. ACM, Oct. 2008, pp. 459–468.

[15] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P.-Y. Strub, "Implementing TLS with verified cryptographic security," in *IEEE Symposium on Security & Privacy*, 2013, pp. 445–462.

[16] B. Blanchet, "Proverif users," Available at http://prosecco.inria.fr/personal/bblanche/proverif/proverif-users.html.

[17] ——, "Computationally sound mechanized proofs of correspondence assertions," in *20th IEEE Computer Security Foundations Symposium (CSF'07)*. IEEE, Jul. 2007, pp. 97–111, extended version available as ePrint Report 2007/128, http://eprint.iacr.org/2007/128.

[18] ——, "A computationally sound mechanized prover for security protocols," *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 4, pp. 193–207, Oct.–Dec. 2008.

[19] ——, "Automatic verification of correspondences for security protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 363–434, Jul. 2009.

[20] ——, "Automatically verified mechanized proof of one-encryption key exchange," in *25th IEEE Computer Security Foundations Symposium (CSF'12)*. IEEE, Jun. 2012, pp. 325–339.

[21] ——, "Modeling and verifying security protocols with the applied pi calculus and ProVerif," *Foundations and Trends in Privacy and Security*, vol. 1, no. 1–2, pp. 1–135, Oct. 2016.

[22] ——, "Symbolic and computational mechanized verification of the ARINC823 avionic protocols," Inria, Research Report RR-9072, May 2017, available at https://hal.inria.fr/hal-01527671.

[23] B. Blanchet, A. D. Jaggard, A. Scedrov, and J.-K. Tsay, "Computationally sound mechanized proofs for basic and public-key Kerberos," in *ACM Symposium on Information, Computer and Communications Security (ASIACCS'08)*. ACM, Mar. 2008, pp. 87–99.

[24] D. Boneh, E. Shen, and B. Waters, "Strongly unforgeable signatures based on computational Diffie-Hellman," in *Public Key Cryptography - PKC 2006*, ser. LNCS, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., vol. 3958. Springer, Apr. 2006, pp. 229–240.

[25] D. Cadé and B. Blanchet, "From computationally-proved protocol specifications to implementations and application to SSH," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, vol. 4, no. 1, pp. 4–31, Mar. 2013.

[26] "Application security solution for the aeronautical communications network with encryption support," Document ATNP SB3W0411: Certicom presentation to ICAO ATN Panel, Working Group B, Subgroup B3, Jan. 2002.

[27] L. Chen and Q. Tang, "Bilateral unknown key-share attacks in key agreement protocols," *Journal of Universal Computer Science*, vol. 14, no. 3, pp. 416–440, Feb. 2008.

[28] C. J. Cremers, "Unbounded verification, falsification, and characterization of security protocols by pattern refinement," in *15th ACM conference on Computer and Communications Security (CCS'08)*. ACM, 2008, pp. 119–128.

[29] I. B. Damgård, "A design principle for hash functions," in *Advances in Cryptology–CRYPTO89 Proceedings*, ser. LNCS, G. Brassard, Ed., vol. 435. Springer, Aug. 1989, pp. 416–427.

[30] W. Diffie, P. C. van Oorschot, and M. J. Wiener, "Authentication and authenticated key exchanges," *Designs, Codes and Cryptography*, vol. 2, no. 2, pp. 107–125, Jun. 1992.

[31] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. IT-29, no. 12, pp. 198–208, Mar. 1983.

[32] T. Frosch, C. Mainka, C. Bader, F. Bergsma, J. Schwenk, and T. Holz, "How secure is TextSecure?" in *IEEE European Symposium on Security and Privacy (Euro S&P)*, 2016, pp. 457–472.

[33] S. Goldwasser, S. Micali, and R. Rivest, "A digital signature scheme secure against adaptative chosen-message attacks," *SIAM Journal of Computing*, vol. 17, no. 2, pp. 281–308, Apr. 1988.

[34] "ICAO doc 9880 part IV: Manual on detailed technical specifications for the Aeronautical Telecommunication Network (ATN) using ISO/OSI standards and protocols, part IV — directory services, security and systems management," 2010, first edition. International Civil Aviation Organization.

[35] "ICAO doc 9896: Manual on the Aeronautical Telecommunication Network (ATN) using Internet Protocol Suite (IPS) standards and protocols," 2010, first edition. International Civil Aviation Organization.

[36] N. Kobeissi, K. Bhargavan, and B. Blanchet, "Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach," in *2nd IEEE European Symposium on Security and Privacy (EuroS&P'17)*. IEEE, Apr. 2017, pp. 435–450.

[37] G. Lowe, "Breaking and fixing the Needham-Schroeder public-key protocol using FDR," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, vol. 1055. Springer, 1996, pp. 147–166.

[38] M. Luby and C. Rackoff, "How to construct pseudorandom permutations from pseudorandom functions," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 373–386, 1988.

[39] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, Dec. 1978.

[40] NIST, "SP 800-38A: Recommendation for block cipher modes of operation, methods and techniques," Available at http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf, 2001.

[41] ——, "FIPS 180-2: Secure hash standard," Available at http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf, Aug. 2002.

[42] ——, "FIPS 198: The keyed-hash message authentication code (HMAC)," Available at http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf, Jul. 2008.

[43] T. Okamoto and D. Pointcheval, "The gap-problems: a new class of problems for the security of cryptographic schemes," in *International Workshop on Practice and Theory in Public Key Cryptography (PKC'2001)*, ser. LNCS, K. Kim, Ed., vol. 1992. Springer, Feb. 2001, pp. 104–118.

[44] A. Pironti, P.-Y. Strub, and K. Bhargavan, "Identifying website users by TLS traffic analysis: New attacks and effective countermeasures," INRIA Paris Rocquencourt, Tech. Rep. 8067, 2012, available at https://hal.inria.fr/hal-00732449/.

[45] B. Schmidt, S. Meier, C. Cremers, and D. Basin, "Automated analysis of Diffie-Hellman protocols and advanced security properties," in *25th IEEE Computer Security Foundations Symposium (CSF'12)*. IEEE Computer Society, Jun. 2012, pp. 78–94.

[46] J.-K. Tsay and S. F. Mjølsnes, "A vulnerability in the UMTS and LTE authentication and key agreement protocols," in *Computer Network Security (MMM-ACNS'12)*, ser. LNCS, I. Kotenko and V. Skormin, Eds., vol. 7531. Springer, Oct. 2012, pp. 65–76.

[47] P. Wuille, "Dealing with malleability," https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki, Mar. 2014.

[48] A. C. Yao, "Theory and applications of trapdoor functions," in *23rd Annual Symposium on Foundations of Computer Science (FOCS'82)*. IEEE Computer Society, Nov. 1982, pp. 80–91.