Coauthentication

Jay Ligatti Cagri Cetin Shamaria Engram Jean-Baptiste Subils Dmitry Goldgof
Department of Computer Science and Engineering
University of South Florida

Technical Report CSE-SEC-092418

ABSTRACT

This paper introduces and evaluates collaborative authentication, or coauthentication, a single-factor technique in which multiple registered devices work together to authenticate a user. Coauthentication provides security benefits similar to those of multi-factor techniques, such as mitigating theft of any one authentication secret, without some of the inconveniences of multi-factor techniques, such as having to enter passwords or biometrics. Coauthentication provides additional security benefits, including: preventing phishing, replay, and man-in-the-middle attacks; basing authentications on high-entropy secrets that can be generated and updated automatically; and availability protections against, for example, device misplacement and denial-of-service attacks. Coauthentication is amenable to many applications, including *m*-out-of-*n*, continuous, group, shared-device, and anonymous authentications. The principal security properties of coauthentication have been formally verified in ProVerif, and implementations have performed efficiently compared to password-based authentication.

ACM Reference Format:
Jay Ligatti Cagri Cetin Shamaria Engram Jean-Baptiste Subils Dmitry Goldgof. 2018. Coauthentication. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Authentication is one of the most common security activities end-users perform. Authentication is also a common target of attacks, through phishing, guessing, man-in-the-middle, token-theft, and related vectors. Due to the commonality of using and attacking authentication systems, even modest improvements to their security or usability may produce significant benefits.

1.1 Background

As is well understood, user authentication is based on factors, the three standard factors being what you know (human-entered secrets like passwords), what you have (physical tokens like keys, electronic remote controls, or smartcards), and what you are (biometrics like fingerprints). Every authentication system, regardless of the factors used, is based on secrets, which could take the form of passwords, patterns of metallic teeth on keys, radio frequencies at which devices transmit data, codes stored on devices and transmitted, fingerprints, etc. Authentication systems aim to protect against attackers who have not obtained the required secrets.

Each authentication factor has advantages and disadvantages [25]. For example, tokens are susceptible to theft, but doing so in the obvious way requires physical access. Users will often notice physical theft of a token more readily than a remote theft or guessing of a password or biometrics. However, tokens have traditionally relied on special-purpose hardware and consequently been more expensive to implement and deploy than other factors. In addition, usability benefits of tokens have traditionally been offset by the costs of having to carry and handle the tokens [25, 34].

Multi-factor authentication attempts to improve security by requiring successful attacks to compromise every factor being used. One popular two-factor mechanism combines a username/password with a second password (a one-time password, OTP) texted to the user's phone [14]. Alternatively, instead of receiving an OTP from the authenticator, the phone may share a cryptographic key with the authenticator and generate its own OTP, called a time-based OTP or TOTP, as a cryptographic hash, using the shared key, of the current time [23]. A benefit of such mechanisms is that the physical-token factor is a device already possessed and carried by the user, thus avoiding expensive, dedicated hardware.

However, multi-factor techniques add the inconveniences of each factor required. For example, because OTP and TOTP techniques require users to enter two passwords and carry a registered device, they suffer from the nontrivial usability drawbacks of password-based authentication mechanisms (e.g., [12, 15, 21, 27, 28]) and the inconvenience of having to access a mobile device to authenticate.

This latter inconvenience, of having to access one's registered mobile device to authenticate, has lessened over time, as the overwhelming majority of adults have gone from having zero personal smart devices accessible at all times to having one personal smart device—a smartphone—accessible at all times [13].

With the growth of the Internet of Things, ubiquitous computing, and wearable, edible, and implantable devices, the overwhelming majority of adults will soon have multiple personal smart devices accessible at all times, all of which can be registered and used to authenticate. For example, to log in to a website, open a door, or start an engine, *two* of a user's registered devices, perhaps a smartphone and smartwatch, might participate in the authentication. A gate or garage door might authenticate a request to open by requiring participation from both a registered car and a registered smartphone; then stealing only the car, or only the phone, would be insufficient for opening the door.

Even today many people only authenticate to certain services when multiple of their devices are present. For example, a user *U*

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

may log in to banking services only from a certain PC while in the presence of U's smartphone. In this case the banking service could register these two user devices to U and require their participation in every authentication of U. Because the PC and smartphone are separate and heterogeneous, successfully stealing or otherwise attacking one device does not imply a successful attack on the other device. It is therefore of value to protect against attacks on only one of the two user devices.

We call this single-factor technique, in which multiple devices collaborate to authenticate a user, *coauthentication*. The user devices collaborate through cryptographic protocols, such that an authenticator receives message(s) proving that all required user devices approve the authentication. Attackers who steal only one of the user devices cannot authenticate, because the unstolen device will not approve the authentication.

Benefits of coauthentication include protecting against the compromise of authentication secrets (cryptographic keys); preventing phishing, replay, and man-in-the-middle attacks; basing authentication on high-entropy secrets that can be generated and updated automatically; avoiding the inconveniences of factors like passwords and biometrics; implementing advanced authentication functionalities, including m-out-of-n, continuous, group, shared-device, and anonymous authentication; and, when implementing m-out-of-n authentication, providing availability protections against device misplacement and denial-of-service attacks.

1.2 Contributions and Roadmap

As far as we are aware, coauthentication is the first single-factor, multi-device technique for authenticating users without passwords or biometrics.

This paper introduces and evaluates coauthentication, including several specific coauthentication system designs, protocols, and implementations. It makes the following contributions.

- Example coauthentication system designs, attack models, policies, and applications are presented (Section 2).
- Coauthentication protocols, having strong two-way authentication and forward-secrecy properties, are defined (Section 3).
- The principal security properties of the coauthentication protocols are formally verified, using ProVerif [3, 4], under a small set of explicitly stated, realistic assumptions (Section 4).
- The implementability and performance of the coauthentication protocols are evaluated (Section 5).
- Several extensions and generalizations of coauthentication are provided (Section 6).
- In a discussion of related work, it is shown that existing authentication techniques, specifically those like OTPs that may involve multiple user devices, can also benefit from the coauthentication protocols (Section 7).

Section 8 concludes and describes ongoing work.

2 COAUTHENTICATION SYSTEM DESIGNS, POLICIES, AND APPLICATIONS

The devices involved in coauthentication are the *authenticator* (e.g., a server deciding whether to authenticate a user), the *requestor* (on which the current authentication attempt is initiated), and one or more *collaborators*. The requestor and collaborator(s) are *registered* with the authenticator, meaning that the devices have access to a secret that the authenticator can use to verify the devices' participation in an authentication. This secret accessible to the requestor and collaborator(s) may, for example, be a secret key shared with the authenticator, or a private key K such that the authenticator can verify signatures created with K.

In some coauthentication protocols, the authenticator, upon receiving an authentication *request*, issues one or more *challenges* and awaits one or more valid *responses* to the challenges. Other protocols avoid authentication challenges. In all cases, the authenticator verifies that multiple registered devices, more specifically the secret keys accessible to those devices, participate in the authentication.

2.1 Attack Models and Assumptions

Coauthentication, like multi-factor techniques, protects against theft of any one authentication secret. The secrets in coauthentication are cryptographic keys. Theft of coauthentication secrets may occur in any way, including by remotely compromising devices to obtain their stored keys or physically stealing devices.

Attackers are assumed to be active and can eavesdrop on, insert, delete, and modify communications. Attackers may mount replay and man-in-the-middle attacks.

Attackers are however assumed to be incapable of cryptanalysis; attackers can only infer plaintexts from ciphertexts when also having the required secret key. Without such an assumption, attackers could extract credentials like session keys simply by monitoring and cryptanalyzing legitimate authentications.

Some coauthentication protocols protect against attackers who know all the secrets stored on a device that the victim user possesses. We call such attacks *key-duplication attacks*. For example, an attacker may duplicate a device's secret keys by remotely compromising the device. Alternatively, the attacker may physically steal a device, duplicate all keys accessible to the device, and return the device to the victim user, who may be unaware of the theft and duplication.

To protect against key-duplication attacks, the coauthentication protocols assume that a private communication channel, inaccessible to attackers, exists between the requestor and collaborator devices. Such an assumption is necessary because the duplicated keys must be updated through some channel inaccessible to the attacker; otherwise, the attacker—who has all of the victim device D's keys—could decrypt and obtain any updated keys sent to D, and modify any updated keys sent from D. Private channels may be implemented with short-range communications, such as NFC, zigbee, wireless USB, infrared, or near-field magnetic induction, under the assumption that attackers cannot access such communications because they are on direct, device-to-device channels. Although this paper assumes that private channels are inaccessible to attackers, practical implementations may encrypt messages on such channels for an added layer of protection, for example against attackers who manage to eavesdrop on a supposedly private channel.

Other coauthentication protocols do not require a private channel between requestor and collaborator devices. Although these protocols do not protect against key-duplication attacks, they do

protect against attackers who obtain keys by stealing devices (without duplicating the keys in, and returning, the devices). In other words, the attack model for these all-public-channel protocols assumes that if an attacker has obtained a device D's authentication secret, then D's legitimate user no longer possesses D.

All of this paper's coauthentication protocols assume that devices in the user's possession run as intended during the coauthentication process. Without such an assumption, malware on the user's requestor device could simply leak decrypted session keys or any other unencrypted private data, and malware on the user's collaborator device could simply approve an attacker's authentication requests. Protecting against malware that is actively running on a device in the user's possession, while the user is authenticating, is beyond the scope of coauthentication.

All of this paper's coauthentication protocols also assume that authenticators run as intended during the coauthentication process. Without such an assumption, malware on the authenticator could simply leak secrets or allow all authentication requests. Protecting against malware on authenticators is beyond the scope of coauthentication.

2.2 Collaboration Policies

Each collaborator may enforce its own policy defining the circumstances under which it participates in a coauthentication.

For example, a collaborator may only participate in an authentication after a user has clicked a button or provided some other input to confirm participation. Under this policy, if an attacker steals or compromises the requestor and initiates a coauthentication, the legitimate user will not confirm the attacker-initiated authentication on the collaborator, so the authentication attempt will fail.

Alternatively, a collaborator may automatically participate in an authentication but warn the user, or log, that it has done so, for example by displaying a text alert with an audible warning sound (e.g., a text message). The alert could provide a simple interface for the user to notify the authenticator if the collaboration was unauthorized (i.e., an attacker-initiated authentication).

The first of these example policies, which we call the *disallow-by-default* collaboration policy, only collaborates when a user confirms the authentication. The second policy, which we call the *allow-by-default-with-warning* collaboration policy, relies on users to observe a warning and handle unauthorized collaborations after the fact. For many applications the usability benefits of the allow-by-default-with-warning policy may outweigh the security costs; many modern authentication systems email or text users after suspicious logins and request after-the-fact notification of unauthorized access.

Additional collaboration policies are possible. For example, a collaborator could decide whether to participate in a coauthentication based on the requestor's proximity, that is, whether the requesting device is co-located with the collaborator. In applications where the attack vector of concern is device theft, a collaborator may presume that a co-located requestor has not been stolen. Such a collaborator may tacitly allow collaborations with co-located requestors but show warnings for, require explicit confirmations for, or disallow entirely, collaborations with non-co-located requestors.

Description of the collaboration policy
Disallow by default (require user confirmation before collaborating)
Allow by default, with a warning or log of the collaboration
If co-located then tacitly allow, else allow by default with a warning
If co-located then tacitly allow, else disallow by default
If co-located then tacitly allow, else disallow entirely
If co-located then allow by default with a warning, else disallow by default
If co-located then allow by default with a warning, else disallow entirely
If co-located then disallow by default, else disallow entirely

Table 1: Example collaboration policies.

Table 1 lists several of these example collaboration policies. Many others are possible, such as taking into account the source of authentication requests or how much time has passed since earlier collaborations.

2.3 Example Applications

Besides the more-obvious applications of authentication, such as logging in to operating systems and web services, authentication occurs in less-obvious, but common, ways, including pushing a button on a remote control to open a door or gate, using a physical key to unlock a door or start an engine, or swiping, scanning, or inserting a payment card, passport, or driver's license.

A thief who successfully breaks into a car containing a garage-door or gate controller can push a button to open the victim's garage or gate. Such attacks have occurred [22, 32]. With coauthentication, a garage-door or gate controller may require both the requestor (a car or a remote control in the car) and a collaborator (a smartphone) to participate in the authentication required for opening. Then an attacker stealing only the victim's car, or only the victim's smartphone, cannot open the door or gate. This benefit is nontrivial due to the *heterogeneity* of the required user devices (car and phone): people rarely leave phones in cars unattended, so a successful attack on, or theft of, one of the two user devices does not normally provide all the secrets required for coauthentication.

Door locks are a similar application. Here the requestor may be a radio transmitter, for example on smart apparel, that sends requests to all locks (authenticators) located within 1m, and the collaborator may be a smartphone. The collaboration policy might require that the phone tacitly allows collaborations for co-located requestors and disallows, with warnings, all other collaborations. Such a system mitigates the recent relay attacks on car doors [24].

Payment cards might also be coauthenticated, to require participation of a registered smartphone as a collaborator to the requesting payment card (which itself might be combined into an existing device, such as a smartwatch). This coauthenticated payment system would achieve security protections similar to systems in which an OTP, sent to the user's phone, is required for payment-card use; however, the coauthentication system could run automatically, without requiring the user to enter any passwords, for example with the smartphone enforcing the second-to-last collaboration policy listed in Table 1.

When run automatically, without requiring user interaction, coauthentication is a zero-interaction authentication system [8]. By making authentications transparent and unobtrusive, zero-interaction

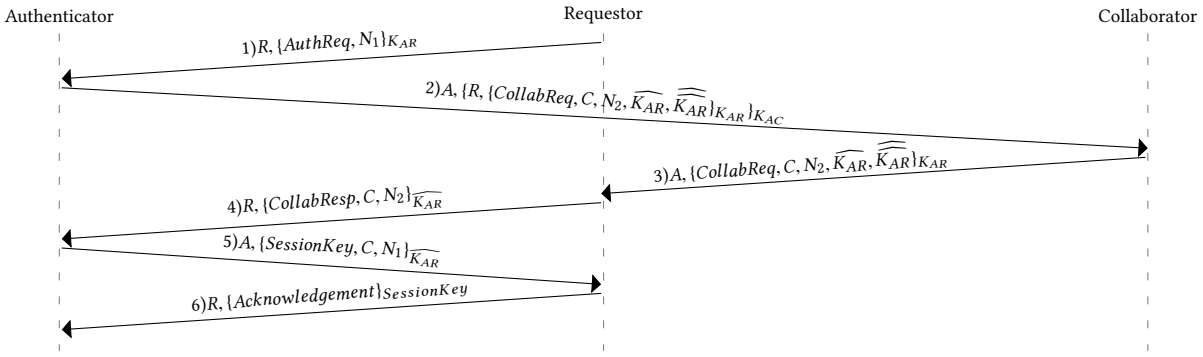

Figure 1: The full coauthentication protocol. Secret key K_{AR} (K_{AC}) is shared between authenticator and requestor (collaborator). Each N_i is a nonce, and $\{M\}_K$ is the encryption of M using key K. The third message is sent through a private channel.

systems enable more devices to benefit from authentication, without fatiguing users with authentication activities. These benefits include enforcing access controls and adjusting to the preferences of each registered user. For example, smart home assistants, smart appliances, and computer components (microphones, keyboards, cameras, memory modules, ALUs) may coauthenticate users to mitigate unauthorized use; televisions may coauthenticate users to enforce parental controls; and chairs, lights, HVAC systems, etc., may coauthenticate users to adjust to their personal preferences.

For this same reason of coauthentication being able to run transparently, or with limited user interaction such as clicking a confirmation button, coauthentication is well suited to continuous authentication [1, 8]. Because continuous authentication requires reauthenticating users periodically, only zero- and low-interaction techniques are appropriate for this application.

3 THE FULL COAUTHENTICATION PROTOCOL

Figure 1 illustrates the full coauthentication protocol for two user devices. Authentication secrets in this protocol are shared symmetric-cryptography keys, and there is only one collaborator.

Following the flow of data in Figure 1, the full protocol operates as follows. Assume that during device registration, the authenticator A and requestor R share a secret key K_{AR}, and the authenticator A and collaborator C share a secret key K_{AC}.

(1) Requestor R initiates the coauthentication by sending the authenticator A its ID and an encrypted authentication-request message containing a challenge nonce N_1 (which serves to authenticate A to R).
(2) Authenticator A receives and decrypts the request message, finds that the requestor R is registered to a user having collaborating device C, creates a challenge nonce N_2 (which serves to authenticate R to A), generates *two new keys* ($\widehat{K_{AR}}$ and $\widehat{\widehat{K_{AR}}}$) to share with R (to rotate keys, to ensure forward secrecy and prevent key-duplication attacks), and double encrypts these data in a collaboration-request message to C, the first (inner) encryption using K_{AR} and the second (outer) encryption using K_{AC}. By double encrypting nonce N_2, the authenticator ensures participation of both user devices' secret keys (K_{AR} and K_{AC}) in the coauthentication.
(3) Collaborator C receives and decrypts the previous message, verifies the identity of the requestor, and forwards the decrypted message (which is still ciphertext encrypted with K_{AR}) to requestor R through a private channel.
(4) Requestor R receives and decrypts this message using K_{AR}, verifies the identity of the collaborator, and obtains N_2, $\widehat{K_{AR}}$, and $\widehat{\widehat{K_{AR}}}$. The requestor then generates and sends the authenticator a collaboration-response message containing N_2 encrypted with its first updated key, $\widehat{K_{AR}}$. The requestor saves the second updated key, $\widehat{\widehat{K_{AR}}}$, for a future coauthentication request.
(5) Authenticator A receives the collaboration-response message, decrypts, and verifies the collaborator's identity and that the received nonce matches the N_2 it sent earlier. Because A has now verified participation of both keys K_{AR} and K_{AC}, it sends an authentication-complete message, for example containing a session key, to the requestor R.
(6) Requestor R sends an acknowledgment to the authenticator.

Timestamps may be added to these messages, for example to implement timeouts or fine-grained logging.

Notice that full coauthentication stores three keys long term: K_{AR} may be stored long term before the current round of authentication, $\widehat{\widehat{K_{AR}}}$ may be stored long term after the current round of authentication, and K_{AC} may be stored long term before and after the current round of authentication.

3.1 Properties of the Full Protocol

The full coauthentication protocol uses nonces to authenticate the requestor and authenticator to each other—session keys are only shared between mutually authenticated devices. Requestor R only shares session keys with authenticated As, and authenticator A only shares session keys with authenticated Rs.

The full protocol also employs key rotation to ensure forward secrecy. An attacker who acquires the keys stored long term on at most one user device cannot obtain past session keys. Each session key is encrypted with an updated $\widehat{K_{AR}}$.

The full protocol mitigates man-in-the-middle attacks by making the authentication secrets shared between the authenticator and

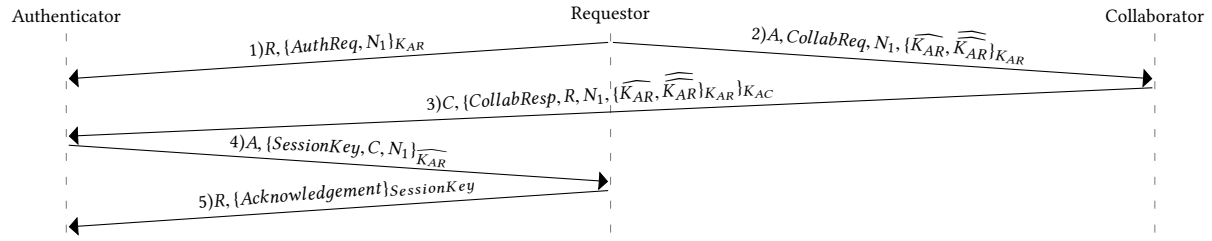

Figure 2: A coauthentication protocol omitting authenticator challenges. The second message is sent through a private channel.

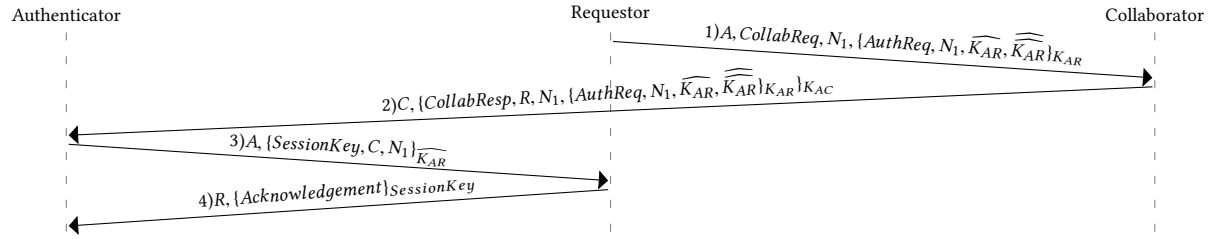

Figure 3: A challengeless coauthentication protocol incorporating message forwarding. The first message is sent through a private channel.

user devices be cryptographic keys, used to encrypt communications. In contrast, man-in-the-middle attacks may be possible on password or biometrics systems because the authenticator may only share, with users or user devices, secrets that are insufficient for cryptographic use. For example, a man-in-the-middle attack on an OTP system may proceed as follows: the victim enters a username and password on a fake website; the fake website forwards this information to the real website, which then issues an OTP; the victim receives and enters the OTP into the fake website; the attacker completes the authentication on the real website and masquerades as the user. In this case the shared username/password (or hash thereof) is insufficient for providing the cryptographic properties needed to mitigate man-in-the-middle attacks.

Now suppose an attacker acquires the long-term secrets stored on at most one user device. Acquiring K_{AC} only enables an attacker, even one with access to the private channel, to permit or deny authentications initiated by the victim. Attackers are already assumed to be active and consequently capable of denying service by dropping network messages. Acquiring K_{AC} therefore provides an attacker with no new capabilities (and Section 6.5 describes extensions of coauthentication that mitigate denial-of-service attacks on user devices).

On the other hand, acquiring only the K_{AR} to be used in the next coauthentication request enables an attacker to request authentication, but assuming an appropriate collaboration policy, the collaborator will notify the victim user of the authentication attempt. From the victim's perspective, this attacker-initiated authentication attempt will be unexpected, so the victim will deny collaboration and therefore the authentication.

Acquiring only the K_{AR} to be used in the next coauthentication request also enables an attacker mounting a key-duplication attack to wait for and decrypt a legitimate authentication request coming from the requestor device, still in the victim's possession. However, such an attacker only obtains nonce N_1 in the process and cannot decrypt any of the remaining messages in the protocol, because they are either encrypted with different keys or sent on a private channel. Obtaining K_{AR} and N_1 provides an attacker with no new capabilities.

The full coauthentication protocol therefore protects against attackers who have acquired the long-term secrets stored on at most one user device. ProVerif has been used to formalize and verify these arguments, as described in Section 5.

3.2 Variation: Omitting the Challenge-Response

It is possible to avoid the challenge-response portion of the full coauthentication protocol, implemented with nonce N_2, by having the requestor send two requests, one to the authenticator (to request authentication) and another to the collaborator (to request collaboration).

Figure 2 shows such a challengeless protocol. The requestor sends two requests, one to the authenticator and another to the collaborator, containing the same nonce N_1. The requestor also includes the updated versions of K_{AR} in its collaboration-request message, which the collaborator forwards to the authenticator. After verifying that both the requestor and its registered collaborator have participated in an authentication by sending the same N_1, the authenticator sends a new session key to the requestor, encrypted with the proper updated version of K_{AR}. As in the full protocol, this challengeless version results in the authenticator and requestor sharing an updated $\widehat{\widehat{K_{AR}}}$, usable in a subsequent run of the protocol as the new version of K_{AR}.

Having formally verified both the full and challengeless coauthentication protocols, to our knowledge they provide the same security guarantees. The known tradeoffs between these protocols relate to performance. The challengeless protocol is expected to be more efficient overall, due to the omission of challenge creation and the parallelization or batching of some of the communications

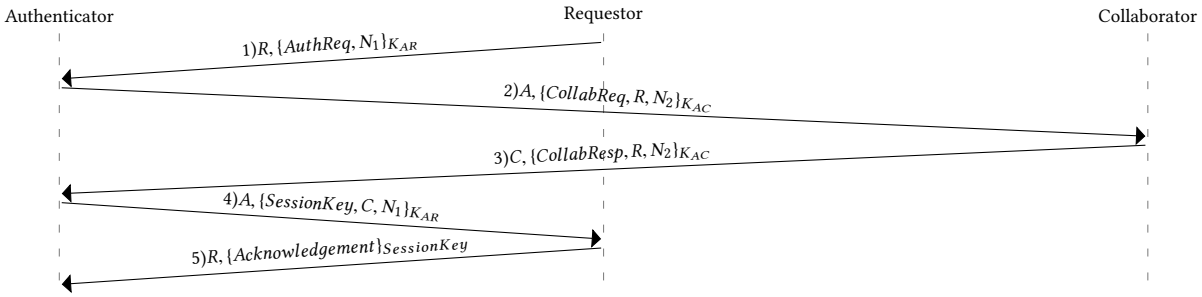

Figure 4: An all-public-channel variation of the full coauthentication protocol (Figure 1).

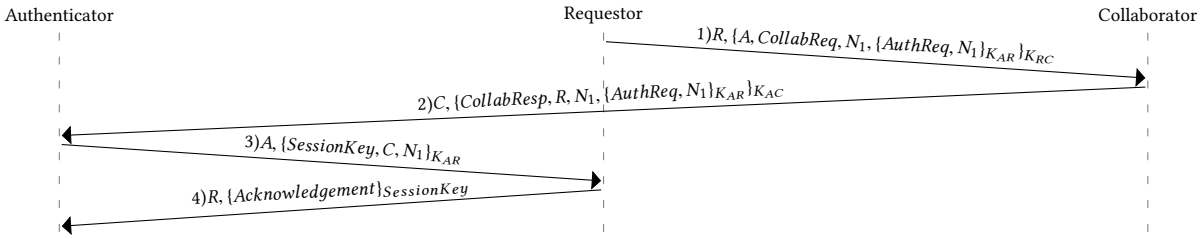

Figure 5: An all-public-channel variation of the challengeless coauthentication protocol with message forwarding (Figure 3).

(e.g., the first and second messages in Figure 2). However, the computations performed by individual devices may be more efficient in the full version. For example, from the requestor's perspective, the challengeless protocol essentially replaces the computations needed to decrypt the third message and generate the fourth message of Figure 1 with the computations needed to generate the second message of Figure 2, including generating updated versions of K_{AR}. For some user devices, such as IoT devices with limited resources, some of these computations may be more expensive than others, making one protocol more efficient than another for those devices.

3.3 Variation: Incorporating Message Forwarding

Figure 3 shows a variation of the challengeless protocol that incorporates message forwarding. The protocol shown in Figure 3 is the same as the one shown in Figure 2 but with the collaborator forwarding the authentication-request message to the authenticator on behalf of the requestor.

3.4 Variation: No Private Channels

In cases where a private channel does not exist between the requestor and collaborator, coauthentication protocols cannot prevent key-duplication attacks. The ability of an attacker, who has acquired all the secrets stored on a user-possessed requestor R, to eavesdrop on and modify all communications to and from R, makes it impossible to update R's secrets without the attacker also obtaining any updates sent to R, and modifying any updates sent from R.

In practice it may be acceptable to dismiss key-duplication attacks by relying on alternative mechanisms to mitigate them. For example, a device's long-term, rarely updated key K_{AR} may be stored in a trusted platform module (TPM) [17]. With K_{AR} in a TPM, we might assume that attackers, who possibly have physical access to the requestor R, may be able to *use* K_{AR} to initiate authentications on R, but cannot *extract* K_{AR} from R. That is, mechanisms like TPMs may mitigate key-duplication attacks by allowing authentication secrets to be used but not extracted, and therefore not duplicated.

It may also be acceptable to dismiss key-duplication attacks in cases where the threat is considered remote or private channels simply cannot be implemented or would be costly to implement.

In any of these cases, the coauthentication protocols can be varied to no longer require a private channel between the requestor and collaborator, yet still protect against non-key-duplication attacks. The attack model for these all-public-channel protocols assumes that if an attacker has obtained a device D's authentication secret, then D's legitimate user no longer possesses D. This attack model still covers attacks based on stealing devices and attempting to authenticate on the stolen devices.

Figures 4–5 show all-public-channel variations of Figures 1 and 3. These all-public-channel protocols match the private-channel protocols, except they abandon all messages and data whose purpose was to update keys (such as the third message in Figure 1) and encrypt any messages sent between the requestor and collaborator with a shared key K_{RC}.

The all-public-channel protocols are overall simpler, and expected to run more efficiently, than the private-channel protocols. The all-public-channel protocols however do not protect against key-duplication attacks and do not satisfy forward secrecy.

In practice a hybrid approach may be preferred: coauthentication keys may be updated only periodically, using private channels at opportune times, while public-channel protocols are used in the common case.

To make an analogy with password-based authentication systems, ideally—from a security perspective—users would update their

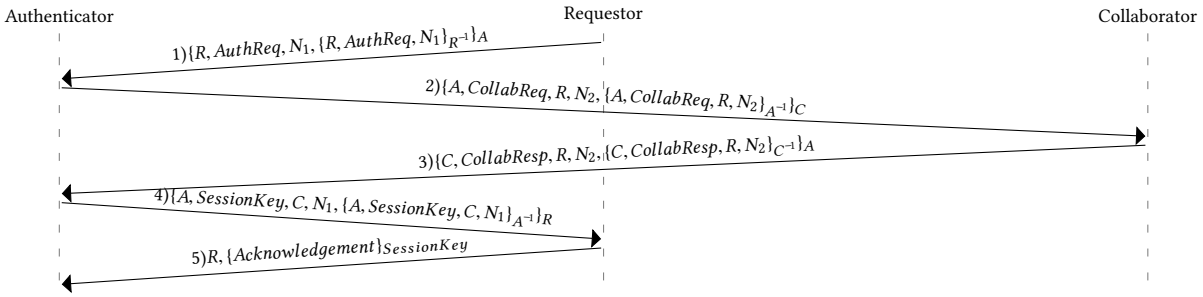

Figure 6: A public-key variation of the protocol in Figure 4. The encryption of M using the requestor's public key is notated $\{M\}_R$, and $\{M\}_{R^{-1}}$ refers to R's digital signature of M (and similarly for authenticator A and collaborator C).

passwords on every authentication, to limit attackers who have acquired passwords. Doing so would be like using the private-channel protocols for coauthentication. In practice, however, security-performance and security-usability tradeoffs are made, and passwords are typically updated only rarely (and unlike coauthentication, through a manual process) [12].

3.5 Variation: Asymmetric Cryptography

Asymmetric (public-key) operations may replace the symmetric-cryptographic operations in coauthentication protocols. Benefits may include using existing public-key infrastructure.

For example, Figure 6 shows a public-key version of the all-public-channel protocol shown in Figure 4. Converting from *all-public*-channel protocols based on symmetric cryptography (such as are shown in Figures 4–5) to ones based on asymmetric cryptography requires only standard techniques (encryptions in the symmetric version based on shared keys are replaced by encryptions in the asymmetric version based on the recipient's public key, digital signatures are added to messages, etc).

Converting from *private*-channel protocols based on symmetric cryptography (such as are shown in Figures 1–3), to ones based on asymmetric cryptography requires additional techniques. In these cases, where defense against key-duplication attacks and forward secrecy need to be provided, the requestor's (public, private) key pair must be updated on every authentication.

Updates to the requestor's (public, private) keys may occur in various ways. A basic design would have the authenticator provide two new key pairs to the requestor in the same ways that the symmetric designs (e.g., Figure 1) have the authenticator provide two new shared keys to requestor. However, having the authenticator generate private keys for the requestor, even private keys only used for the authenticator's services, may not be considered appropriate for asymmetric systems. Less practically, the requestor could update its keys in the public-key infrastructure before or during every authentication.

4 FORMAL EVALUATION

The principal security properties of the example coauthentication protocols shown in Figures 1–6 have been formally verified with ProVerif [3, 4]. ProVerif uses a resolution-based strategy to verify that protocols satisfy desired security properties. A benefit of using ProVerif is that it can model arbitrarily many sessions of a protocol running concurrently.

Our ProVerif encodings of the coauthentication protocols, and the properties verified, are available online [7]. The protocol encodings faithfully follow the communications shown in Figures 1–6.

4.1 Assumptions

The protocols were modeled and verified under the assumptions stated in Section 2.1. The private-channel protocols (Figures 1–3) have strong attack models allowing key-duplication attacks.

The all-public-channel protocols (Figures 4–6) have weaker attack models that assume authentication secrets—K_{AR}, K_{AC}, and K_{RC}—are only accessible to attackers through device theft. In terms of the ProVerif encodings, this weaker attack model for the all-public-channel protocols means that, in cases where attackers are assumed to know K_{AR}, the collaborator does not respond to collaboration requests. The justification is that if an attacker has acquired K_{AR}, then by assumption the legitimate user does not possess the requestor, so collaboration requests must be for unauthorized, attacker-initiated authentications. It is assumed that, with appropriate collaboration policies, users do not approve collaborations for unauthorized authentications.

In all the protocols, attackers are active and may freely eavesdrop on, insert, delete, and modify communications. Attackers are not constrained to operate according to any of the protocols.

In addition to arbitrary active attackers, each protocol session runs 3 processes (authenticator A, requestor R, and collaborator C), and the main ProVerif process considers arbitrarily many sessions of a protocol running concurrently.

4.2 Verification Setup

Each protocol was verified in 3 runs.

(1) The first run began with attackers knowing no secret keys.
(2) The second run began with attackers knowing all the long-term keys accessible to the collaborator. For the protocols shown in Figures 1–4, attackers were given K_{AC}; for the protocol shown in Figure 5, attackers were given K_{AC} and K_{RC}; and for the protocol shown in Figure 6, attackers were given C^{-1}.
(3) The third run began with attackers knowing all the long-term keys accessible to the requestor. For the protocols shown in Figures 1–3, attackers were given K_{AR} and $\widehat{\widehat{K_{AR}}}$; for the protocol shown in Figure 4, attackers were given K_{AR}; for the protocol shown in Figure 5, attackers were given K_{AR}

and K_{RC}; and for the protocol shown in Figure 6, attackers were given R^{-1}.

In all 3 runs of each of the 6 protocols, we attempted to verify the following 4 security properties.

P1: Secrecy of the session key. The session key is only known to the authenticator and requestor. This property subsumes forward secrecy in the third run of the private-channel protocols (Figures 1–3) because knowing the requestor's future authentication secret ($\widehat{\widehat{K_{AR}}}$, which becomes K_{AR} in the next round of authentication) does not leak session keys.

P2: Secrecy of the acknowledgment. The acknowledgment data is only known to the authenticator and requestor.

P3: Authentication of R to A. With one exception, we specified authentication of R to A as requiring that if the authenticator receives an acknowledgment of a session key (and therefore believes it shares the session key with the requestor) then the requestor was indeed its interlocutor and the collaborator indeed collaborated. This is an event-based property [35] having the form

$$endA \implies (beginA \wedge collabA),$$

where *endA* refers to the event of A receiving the acknowledgment, *beginA* to R sending the authentication request, and *collabA* to C sending its participation message (in the third message of Figures 1, 2, 4, and 6 and the second message of Figures 3 and 5).

The one exception to encoding *P3* in this way is for the second run of the protocols, where the attacker is given K_{AC}. In this case, the attacker may use K_{AC} to collaborate with legitimate authentication requests, thus helping legitimate authentications succeed, which we do not consider an attack. Therefore, for the second run of the protocols, we specify property *P3* as only requiring

$$endA \implies beginA,$$

that is, if the authenticator believes it shares the session key with the requestor then the requestor was indeed its interlocutor (but the attacker, rather than the collaborator, may have collaborated).

P4: Authentication of A to R. This property is symmetric to *P3* and, with one exception, requires that if the requestor sends an acknowledgment of a session key (and therefore believes it shares the session key with the authenticator) then the authenticator was indeed its interlocutor and the collaborator indeed collaborated. This property has the form

$$endR \implies (beginR \wedge collabR),$$

where *endR* refers to R sending the acknowledgment, *beginR* to A receiving the authentication request, and *collabR* to C sending its participation message.

As with *P3*, the one exception to encoding *P4* in this way is for the second run of the protocols, in which case *P4* only requires

$$endR \implies beginR,$$

for the same reason explained for property *P3*.

4.3 Verification Results

ProVerif found no attacks on any of properties *P1–P4* in any runs of any of the protocols. That is, ProVerif did not refute any of *P1–P4* in any runs of any of the protocols.

ProVerif did prove *P1*, *P2*, an *P4* for all 3 runs of all 6 protocols, and it proved *P3* for all 3 runs of half of the 6 protocols—the full coauthentication protocol (Figure 1) and the two protocols having the same message scheme as the full protocol (Figures 4 and 6). It also proved *P3* for the second and third runs of the protocol shown in Figure 5.

For all runs of the protocols shown in Figures 2 and 3, and for the first run of the protocol shown in Figure 5, ProVerif could not prove *P3*. It produces a "possible" attack trace in which a man-in-the-middle sits between A and R, and A and C, and simply collects and forwards all messages sent to and from A. This trace is a false attack because the authenticator completes the protocols with R having sent the original authentication request and C having sent its participation message, despite the fact that the attacker touched these messages while acting as an intermediary.

We also note that these results are for the stronger, injective-correspondence versions of properties *P3* and *P4*. The injective-correspondence versions require there to be a unique predecessor event for each end event [5]; for example, the injective version of *P3* requires that for each *endA* event there exists a unique *beginA* predecessor event. The non-injective versions allow end events to have non-unique predecessor events. ProVerif was able to prove the weaker, non-injective version of property *P3* for all runs of all protocols.

5 EMPIRICAL EVALUATION

We have implemented and measured the performance of full coauthentication (Figure 1) and all the variations shown in Figures 2–6. To establish a baseline of performance, we also implemented and measured the performance of a basic password authentication system. In total, 7 authentication systems were evaluated.

5.1 Implementations

The password-authentication system only uses two devices (requestor and authenticator), while all of the coauthentication systems use three devices (authenticator, requestor, and collaborator).

To make performance comparisons more meaningful, the implementations were uniform to the extent possible. Each authenticator was implemented as a Java server application using Spring Boot [33], and each requestor and collaborator was implemented as an Android application. All nonces were 64-bit strings dynamically generated with Java's cryptographically strong random number generator class `java.security.SecureRandom`. All session keys in authentication-complete messages were 256-bit strings dynamically generated in the same way. The versions of coauthentication shown in Figures 1–3 also required two new keys to be generated dynamically, $\widehat{K_{AR}}$ and $\widehat{\widehat{K_{AR}}}$, again with Java's cryptographically strong random number generator. All other cryptographic keys were hardcoded, with shared keys assumed to have been shared before the implementations began running. All symmetric cryptographic operations were implemented with 256-bit CBC-mode

Implementation	Bytes Transmitted	Application-Layer Time (ms)				Authentication Time (ms)
		Authenticator	Requestor	Collaborator	Total	
Password	3212	0.28	1.50	—	1.80	136
Figure 1	1198	2.58	22.5	18.4	43.5	594
Figure 2	1088	1.36	20.1	20.9	42.4	475
Figure 3	885	1.16	17.2	19.4	37.8	473
Figure 4	835	1.88	6.18	23.5	31.6	142
Figure 5	1075	0.94	14.9	23.3	39.1	131
Figure 6	7158	0.43	2.94	12.9	16.3	388

Table 2: Average performance of the authentication systems over 100 runs.

AES, and all asymmetric cryptographic operations were implemented with HTTPS using 2048-bit RSA and self-signed certificates, through standard `javax.crypto` libraries.

To broadly mimic typical password-authentication systems, we implemented ours to run over HTTPS (again, using 2048-bit RSA and self-signed certificates). The requestor in our implementation sends the authenticator a username and password hardcoded in the requestor, with the username and password each being 8 characters because such length is common [9]. The authenticator receives and decrypts the username and password, adds salt to the password, hashes (with SHA-256), and verifies that the hash matches its hardcoded expected hash for the given username.

The public-key version of coauthentication shown in Figure 6 was also implemented to run over HTTPS configured in the same way. All the coauthentication protocols shown in Figures 1–5 sent all public-channel messages over TCP.

All public-channel messages, in all implementations, were sent through standard Wi-Fi channels. For communicating private-channel messages, that is, messages from the collaborator to the requestor in Figures 1–3, our implementations used Bluetooth, though it has known vulnerabilities [11].

Each run of each implementation opened new network connections, including a new Bluetooth connection in the implementations of Figures 1–3. Connections were never reused between runs of the implementations, and the Android applications were restarted for each run.

5.2 Experimental Setup and Results

The implementations were executed on the following devices. The authenticator was always a MacBook Pro laptop running macOS Sierra version 10.12.6 and having 16GB of memory and a 2.2GHz Intel quad-core i7 processor. Due to the popularity of mobile access to authentication services, the requestor was always a smartphone, a Samsung Galaxy s8 Plus running Android 8.0.0 and having 4GB of memory, a Qualcomm MSM 8998 octa-core (a 2.35GHz quad-core and a 1.9GHz quad-core) processor, and Bluetooth 5. The collaborator was always a Motorola Nexus 6 running Android 7.1.1 and having 3GB of memory, a 2.7GHz quad-core Qualcomm Snapdragon 805 processor, and Bluetooth 4.1.

Each of the implementations was run 100 times, in a uniform environment of normal (workday) university-network usage and standard loads of kernel and user-level applications running.

The following measurements were made for each run:

- The network usage, that is, the number of bytes transmitted over the course of the run. Due to unreliability in the communication channels, the number of bytes transmitted varied with each run. The network usage was measured with Android's standard network-monitoring class `android.net.TrafficStats`.
- The application-layer real time each device consumed. This measurement was made by starting a timer when beginning to process any newly received message or request, stopping the timer when finished preparing a response, taking the difference, and summing all of these times for each device. For example, the application-layer real time consumed by the authenticator in full coauthentication is the sum of the real times it consumes processing the requestor's and collaborator's messages, including generating new keys and a challenge nonce and performing the required encryptions and decryptions. Application-layer times exclude all time spent establishing connections and transmitting messages in the underlying TCP, HTTPS, and Bluetooth protocols.
- The total authentication time. This is the real time, measured on the requestor, from beginning to prepare an authentication request until finishing obtaining a plaintext session key.

Tables 2–3 summarize the results of running each implementation 100 times.

5.3 Performance Analysis

Many of the performance results are as expected. As shown in Table 2, protocols transmitting more or more complex messages, or using HTTPS, transmitted more bytes of data. Network (non-application-layer) activities dominated the performance of all implementations, consuming between 70% and 98.7% of the total authentication time on average.

As shown in Table 3, these network activities also took a highly variable amount of time to complete, over different runs of the same implementation. The coefficients of variation (CVs) for total authentication time ranged up to 53%, indicating high variance. This variance explains the sometimes-substantial differences between median and average total authentication times observed for the same implementation. The coefficients of variation for total application-layer times were substantially smaller for all but the Figure-6 implementation, indicating much less variance.

In terms of application-layer performance, the password system was the most efficient and then the Figure-6 public-key system. Both of these systems benefit, at the application layer, from pushing all the cryptographic operations into the underlying HTTPS layer.

In terms of total authentication time, the Figure-5 system outperformed the others on average, and the outperformance was greater

	Total Application-Layer Time			Total Authentication Time		
Implementation	Average (ms)	Median (ms)	CV	Average (ms)	Median (ms)	CV
Password	1.80	1.80	0.08	136	132	0.23
Figure 1	43.5	42.5	0.23	594	564	0.26
Figure 2	42.4	41.4	0.27	475	430	0.38
Figure 3	37.8	37.4	0.31	473	426	0.45
Figure 4	31.6	30.4	0.22	142	130	0.53
Figure 5	39.1	37.7	0.28	131	93.4	0.49
Figure 6	16.3	13.5	0.33	388	388	0.16

Table 3: Statistics on the performance of the authentication systems over 100 runs. CV refers to the coefficient of variation.

in the median case. The performance of this coauthentication system benefits from transmitting a minimal number of messages over the efficient (relative to HTTPS and Bluetooth) TCP.

Importantly, these performance results exclude human time, though it is known to be substantial for password-based authentication systems. Human entry of a password is expected to take on the order of several seconds [21, 28, 31].

Care should also be exercised when comparing the performance of the password-based system with the performance of the private-channel coauthentication systems (Figures 1–3), which update K_{AR} on every authentication. The advantages of updating K_{AR} are analogous to the advantages of updating a password, so a better comparison would take into account the time required to update passwords. Password update is expected to take on the order of a minute of human time [30], significantly longer than an automatic coauthentication-key update.

We conclude from these results that coauthentication performs efficiently enough to be practical.

6 EXTENSIONS AND GENERALIZATIONS

Extensions and generalizations of coauthentication are possible.

6.1 Additional Challenges and Responses

The full coauthentication protocol uses a challenge-response process in which the authenticator first encrypts a nonce to challenge the requestor R and then encrypts the result to challenge the collaborator C. Receiving a valid response requires participation from, and collaboration between, both R and C.

Many other challenge-response processes are possible. For example, the authenticator may reverse the order of encryptions to require the requestor to participate before the collaborator; the authenticator may only encrypt the challenge with one shared key (e.g., K_{AR}) and wait for a valid response encrypted with the other shared key (e.g., K_{AC}); the authenticator may concurrently send the requestor and collaborator the same or different challenges and require valid responses from both; or the authenticator may issue challenges for which generating valid responses requires interacting with one or more third-party servers. Authenticators may issue challenges requiring other sorts of responses as well, such as requiring message authentication codes (MACs) or passwords or other secrets in responses.

6.2 Locally Broadcasting Challenges

In some applications it may be useful to make some of the coauthentication protocol's communications multicast or broadcast. For example, to require additional user interaction during coauthentication, the requesting device may receive a challenge directly from the authenticator and then display (visually/locally broadcast) the challenge as a QR code [16]. The user could then scan the challenge QR code on the collaborating device, which could then send a valid response to the authenticator.

6.3 Collaboration to Obtain Session Keys

Similar to the double encryption for transmitting challenges in the full coauthentication protocol, authenticators may double encrypt authentication-complete (e.g., session-key) messages. Encrypting session-key messages with both K_{AC} and K_{AR} (or $\widetilde{K_{AR}}$) forces both the requestor and collaborator to participate, to obtain the session key. Requiring this additional collaboration mitigates attacks in which the requestor is noticeably compromised after the collaborator responds to the authentication challenge, giving the collaborator one more chance to confirm the authentication before the requestor obtains the session key.

6.4 Authenticating Other Devices

The coauthentication protocols may also be modified to give the collaborator access to the session key. For example, coauthentication may proceed as normal before the requestor shares the session key with the collaborator, possibly after a mutual authentication between requestor and collaborator. Another example involves a device, pre-authenticated with the requestor, using the requestor as a proxy to authenticate to the authenticator, building a chain of authentication. Such designs enable n-way authentication, or mutual authentication between multiple pairs of devices.

6.5 Multiple Collaborators, m-out-of-n Policies, and Availability Benefits

A variety of coauthentication protocols also exist for cases in which a user has registered more than two devices with an authenticator. The authenticator in such a case may determine some subset of the registered user devices to which to send challenges, possibly based on guidance from the requestor. The authenticator may then send one or more challenges to this subset of devices, such that the challenges cryptographically require participation from some or all of the user devices.

There are advantages to systems in which users register more than two devices with an authenticator. Suppose a user has registered n devices and the authenticator requires any m of the n

devices to coauthenticate, where $2 \leq m \leq n$. In the coauthentication protocols described so far, m=n=2, but now suppose m=2 and n=3. In this case, compromising only one of the user's devices (i.e., obtaining only one device's authentication secrets) is still insufficient for authenticating as that user, because m=2. At the same time, because m<n, the user can be authenticated even after forgetting or losing a device, or having a device become inoperable, for example due to a denial-of-service attack.

This m-out-of-n-device policy, enforced at the authenticator, tolerates the absence of $n-m$ devices. Hence, user-side denial-of-service attacks require denying service to $n-m+1$ devices. When these devices communicate through heterogeneous channels, denial-of-service attacks based on jamming or otherwise interfering with specific communication channels become more difficult to mount.

To prevent attackers from using $n-m$ compromised devices to coauthenticate, m may be further constrained to be greater than $n-m$, that is, $m > n/2$. For example, a system that requires only 2 out of 4 devices to coauthenticate (i.e., m=2=$n/2$) tolerates the absence of 2 devices, but if those 2 devices are absent due to theft, then the thief can use them to coauthenticate. To prevent such attacks, the m-out-of-n-device policy may be constrained to $2 \leq m \leq n < 2m$

The m-out-of-n-device policy can be generalized further, to policies in which devices are, for example, (1) weighted in various ways to get above a threshold (e.g., 2 "votes" are required to authenticate the current user, but each smart shoe only gets half a vote), (2) required (e.g., 2 devices are required but one must be the user's smartphone), or (3) excluded (e.g., high-risk users may not use easily-transferrable smartcards for coauthentication).

6.6 Group Coauthentication

Users may also be coauthenticated simultaneously, as a group. Such authentication subsumes the famous two-person concept for authenticating users who will have access to nuclear and other weapons [10, 36], or to bank vaults. For example, a two-person policy may require two users to simultaneously turn four keys, one in each hand, to gain access to a weapon-deployment system. The goal is to require both users to participate in the authentication.

Because coauthentication requires participation of multiple devices in an authentication, it may require participation of multiple users in an authentication, where each user has at least one registered device. The same coauthentication protocols can be followed to authenticate multiple users' devices simultaneously. More sophisticated group coauthentications could, for example, require participation of m-out-of-n devices from each of j-out-of-k users.

6.7 Device Sharing and Anonymous Coauthentication

Users may also share devices. For example, a garage-door authenticator may receive a request from a shared family car and send challenges to all the smartphones of drivers in the family, or only those smartphones in near-proximity. The smartphones might enforce the collaboration policy of tacitly participating if co-located with the requestor and not participating otherwise.

Alternatively, assume that every collaborator (smartphone) shares *the same* secret key with the garage-door authenticator. Then the authenticator may, upon receiving a request from the family car, respond directly to the car with a challenge requiring participation from any collaborator—and leave it to the car to obtain a collaborator's participation. An interesting aspect of this alternative is the anonymity it provides: the authenticator only communicates with the shared requestor device and does not know which user has been authenticated, nor which device has collaborated. Authentications are still protected against attackers acquiring one of the secret keys.

It is also possible to achieve anonymous coauthentication for systems in which requestor devices are not shared, by having all potential requestors share the same secret key with the authenticator. Because coauthenticators verify usage of keys, anonymity is achieved by having devices share keys.

Of course, these designs only protect anonymity during the authentication process. Authenticators frequently have other opportunities to de-anonymize users, though techniques like onion routing [26] may mitigate some de-anonymizations.

7 ADDITIONAL DISCUSSION OF RELATED WORK

Many existing systems are related to coauthentication.

7.1 Threshold Schemes and Multi-Signatures

An (m, n) threshold scheme enables a secret to be divided among n entities, such that each entity has one piece of the secret and m of the n pieces are required to determine the secret [29]. An (m, n) threshold scheme has cryptographic benefits analogous to the user-authentication benefits of an m-out-of-n-device coauthentication policy; both protect against fewer-than-m entities acting maliciously and at-most-n-minus-m entities being unavailable to participate.

Multi-signature schemes similarly enable different users or devices to generate a joint digital signature [2].

Threshold and multi-signature schemes do not provide coauthentication systems, and vice versa, as they differ in techniques and goals. Threshold (multi-signature) schemes contribute techniques for combining secret-pieces (signatures) into a joint secret (signature), while coauthentication systems require no joint secret or signature. Coauthentication secrets (i.e., keys) may be used only independently, to indicate one device's participation in user-level authentications, without ever being combined. The goals of threshold and multi-signature schemes focus on combining pieces of cryptographic secrets or signatures into joint secrets or signatures, while coauthentication's goals focus on user authentication.

7.2 OTPs and Other Techniques Using Multiple Devices

One group of techniques related to coauthentication uses OTPs, as discussed in Section 1. The standard use of OTPs is as follows. A user enters a username and password on a requestor device, the authenticator SMS-texts an OTP to the user's phone (which may also be the requestor device), and the user sees the OTP and enters it on the requestor device as a second password required for authentication. This use of OTPs differs from coauthentication in several ways, perhaps the most significant being that the OTPs are used in two-factor systems, while coauthentication is a single-factor system. Hence, attackers can break the OTP portion of authentications

by compromising one device, the victim's phone, or by reading the SMS messages sent to the phone [14, 18, 20].

Another related group of techniques use multiple devices to acquire multiple passwords or biometric data [19]. The authenticator combines these data to determine whether to authenticate a user. For example, if a user has a sensor-device implanted in each finger, then each device may send data related to that finger's motion to an authenticator, which can make authentication decisions based on whether a user has moved or gestured in the proper way for that user. Although using multiple devices, this line of work relies on users to enter passwords or biometrics, which are assumed to be unguessable and unforgeable by attackers.

Coauthentication, like other zero- or low-interaction authentication systems [8], shields users from attacks based on guessing or forging authentication secrets, such as password phishing or biometric surveillance. Coauthentication users never have to access or even understand the secrets required for authentication, and coauthentication secrets can be generated automatically, with high entropy, and without concern for whether humans have the resources (cognitive ability, time, etc.) to generate, store, update, or enter the secrets.

Bonneau et al. evaluated authentication techniques, including OTPs, according to three axes: usability, deployability, and security [6]. A total of 25 criteria are considered along these axes, such as whether the techniques require users to memorize secrets or carry devices. As motivated in Section 1, we consider disadvantages related to requiring users to carry devices to be decreasing. In any case, we believe that coauthentication satisfies the majority of Bonneau et al.'s criteria, though it is difficult to make precise claims in this respect, due to subjectivity in the criteria [6, Section V-B]. The most significant criteria coauthentication does not satisfy relate to deployability; deploying coauthentication, like deploying any new authentication technique, would require updating authentication clients and servers, and in some implementations, relying on co-location verification.

7.3 Using Coauthentication Protocols to Implement Existing Multi-Device Techniques

Coauthentication protocols can be used to implement existing multi-device authentication systems.

For example, the full coauthentication protocol shown in Figure 1 can implement OTP-based authentication: the requestor might be a laptop, the initial (static) password might be included in the initial authentication-request message sent from requestor to authenticator, the challenge nonce might be the one-time (dynamic) password, the collaborator might be a smartphone, the communication from authenticator to collaborator might be through SMS, and the communication from collaborator to requestor might occur by displaying the OTP-carrying ciphertext on the smartphone screen and having the user enter it manually on the laptop.

Similarly, the protocol shown in Figure 2 can implement authentication based on biometric data collected from multiple sensors, for example, authentication based on data collected from sensors implanted in fingers [19]. In this case the requestor may request collaboration from multiple sensors, each of which transmits its authentication participation—including motion data collected—to the authenticator. The authenticator collects and considers these participation messages to make authentication decisions.

Implementing existing multi-device authentication systems with coauthentication protocols provides the formally verified security benefits outlined in Section 4. These benefits are sometimes lacking in the existing systems. For example, the protocol shown in Figure 1 provides forward-secrecy properties lacking in many existing authentication systems. In addition, although existing OTP systems are vulnerable to text-message eavesdropping and man-in-the-middle attacks [14, 18, 20], the coauthentication protocols mitigate these attacks.

8 CONCLUSIONS AND ONGOING WORK

The coauthentication protocols and system designs have several potential benefits. Coauthentication:

- protects against compromise of any one authentication secret, similar to multi-factor techniques but without the inconveniences of having to enter passwords (including OTPs) or scan biometrics;
- requires little, and in some implementations no, interaction from users;
- mitigates phishing, replay, and man-in-the-middle attacks (there are no passwords to phish, and the attack models assume active attackers);
- bases authentications on high-entropy secrets that can be generated, exchanged, stored, updated, and used automatically and efficiently (in contrast with password and biometric secrets);
- can implement advanced functionalities, including m-out-of-n, continuous, group, shared-device, and anonymous authentications;
- has formally verified security properties;
- has been implemented and found to perform efficiently enough to be practical;
- can be combined with additional authentication factors;
- provides protocols that may benefit existing multi-device authentication systems, such as those based on OTPs.

Ongoing work is investigating the usability of various authentication mechanisms, including coauthentication. Because coauthentication can be implemented with little-to-no user interaction, we hypothesize that coauthentication mechanisms may have improved usability compared to existing authentication mechanisms, particularly multi-factor mechanisms. Coupling these hypothesized usability benefits with the security benefits outlined in this paper, coauthentication, or a multi-factor authentication with coauthentication as the physical-token factor, may be advantageous for some authentication applications.

REFERENCES

[1] Geneva Belford, Steve Bunch, John Day, Peter Alsberg, Deborah Brown, Enrique Grapa, David Healy, and John Mullen. 1975. *A State-of-the-Art Report on Network Data Management and Related Technology.* Technical Report 150. Center for Advanced Computation, University of Illinois at Urbana-Champaign. Page 132. https://archive.org/details/stateoftheartrep150belf.

[2] Mihir Bellare and Gregory Neven. 2007. Identity-Based Multi-Signatures from RSA. In *Proceedings of the Cryptographers' Track at the RSA Conference.* Springer, 145–162.

[3] Bruno Blanchet. 2001. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proceedings of the IEEE Computer Security Foundations Workshop*. 82–96.

[4] Bruno Blanchet. 2016. ProVerif: Cryptographic protocol verifier in the formal model. http://prosecco.gforge.inria.fr/personal/bblanche/proverif/.

[5] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. 2017. ProVerif 1.98pl1: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial. http://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf.

[6] Joseph Bonneau, Cormac Herley, Paul C van Oorschot, and Frank Stajano. 2012. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *IEEE Symposium on Security and Privacy*. 553–567.

[7] Cagri Cetin and Jay Ligatti. 2018. ProVerif coauthentication files. https://github.com/Coauthentication/FormalModels.

[8] Mark D Corner and Brian D Noble. 2002. Zero-Interaction Authentication. In *Proceedings of the ACM International Conference on Mobile Computing and Networking*. 1–11.

[9] Matteo Dell'Amico, Pietro Michiardi, and Yves Roudier. 2010. Password Strength: An Empirical Analysis. In *Proceedings of IEEE INFOCOM*. 1–9.

[10] Department of Defense 1990. *Nuclear Weapon Accident Response Procedures (NARP)*. Department of Defense. DoD 5100.52-M. https://fas.org/nuke/guide/usa/doctrine/dod/5100-52m/chap15.pdf.

[11] John Dunning. 2010. Taming the Blue Beast: A Survey of Bluetooth-Based Threats. *IEEE Security & Privacy* 8, 2 (2010), 20–27.

[12] Dinei Florencio and Cormac Herley. 2007. A Large-Scale Study of Web Password Habits. In *Proceedings of the International Conference on World Wide Web*. 657–666.

[13] Nancy Gibbs. 2012. Your Life Is Fully Mobile. *TIME* (Aug. 2012). http://techland.time.com/2012/08/16/your-life-is-fully-mobile/.

[14] Paul Grassi, James Fenton, Elaine Newton, Ray Perlner, Andrew Regenscheid, William Burr, Justin Richer, Naomi Lefkovitz, Jamie Danker, Yee-Yin Choong, Kristen Greene, and Mary Theofanos. 2017. NIST Special Publication 800-63B Digital Authentication Guideline. https://doi.org/10.6028/NIST.SP.800-63b.

[15] Philip G Inglesant and M Angela Sasse. 2010. The True Cost of Unusable Password Policies: Password Use in the Wild. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 383–392.

[16] International Standards Organization. 2015. *Information technology – Automatic identification and data capture techniques – QR Code bar code symbology specification*. Technical Report. ISO/IEC 18004:2015. https://www.iso.org/standard/62021.html.

[17] International Standards Organization. 2015. *Information technology – Trusted platform module library – Part 1: Architecture*. Technical Report. ISO/IEC 11889-1:2015. https://www.iso.org/standard/66510.html.

[18] Radhesh Krishnan Konoth, Victor van der Veen, and Herbert Bos. 2016. How anywhere computing just killed your phone-based two-factor authentication. http://fc16.ifca.ai/preproceedings/24_Konoth.pdf.

[19] Tiano Freixas Lopez Lecube, Josh Miller, Thomas Jaeger, Sam Oh, Wenhan Zhao, and Eric Min. 2015. Multi-Device Authentication. US Patent Application 2017/0093846 A1.

[20] Charles McColgan. 2016. Issues with SMS Deprecation rationale. https://github.com/usnistgov/800-63-3/issues/351.

[21] William Melicher, Darya Kurilova, Sean M Segreti, Pranshu Kalvani, Richard Shay, Blase Ur, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Michelle L Mazurek. 2016. Usability and Security of Text Passwords on Mobile Devices. In *Proceedings of the Conference on Human Factors in Computing Systems*. 527–539.

[22] Jorge Milian. 2015. Burglars using stolen garage-door openers in Boynton Beach. http://www.mypalmbeachpost.com/ZU0JluARHfHTpRnGA6A4ZJ/.

[23] David M'Raihi, Salah Machani, Mingliang Pei, and Johan Rydell. 2011. *TOTP: Time-Based One-Time Password Algorithm*. RFC 6238. http://www.rfc-editor.org/rfc/rfc6238.txt.

[24] BBC News Staff. 2017. Mercedes 'relay' box thieves caught on CCTV in Solihull. http://www.bbc.com/news/uk-england-birmingham-42132689.

[25] Lawrence O'Gorman. 2003. Comparing Passwords, Tokens, and Biometrics for User Authentication. *Proc. IEEE* 91, 12 (Dec. 2003), 2021–2040.

[26] Michael Reed, Paul Syverson, and David Goldschlag. 1998. Anonymous Connections and Onion Routing. *IEEE Journal on Selected areas in Communications* 16, 4 (1998), 482–494.

[27] M Angela Sasse, Sacha Brostoff, and Dirk Weirich. 2001. Transforming the 'Weakest Link'–a Human/Computer Interaction Approach to Usable and Effective Security. *BT Technology Journal* 19, 3 (July 2001), 122–131.

[28] Florian Schaub, Ruben Deyhle, and Michael Weber. 2012. Password Entry Usability and Shoulder Surfing Susceptibility on Different Smartphone Platforms. In *Proceedings of the International Conference on Mobile and Ubiquitous Multimedia*. 13:1–13:10.

[29] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (Nov. 1979), 612–613.

[30] Richard Shay, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Alain Forget, Saranga Komanduri, Michelle L Mazurek, William Melicher, Sean M Segreti, and Blase Ur. 2015. A Spoonful of Sugar?: The Impact of Guidance and Feedback on Password-Creation Behavior. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. 2903–2912.

[31] Richard Shay, Saranga Komanduri, Adam L Durity, Phillip Seyoung Huh, Michelle L Mazurek, Sean M Segreti, Blase Ur, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2014. Can Long Passwords Be Secure and Usable?. In *Proceedings of the Conference on Human Factors in Computing Systems*. 2927–2936.

[32] Observer Staff. 2016. Three arrested in Hillsborough County burglary. http://www.plantcityobserver.com/article/three-arrested-hillsborough-county-burglary.

[33] Phillip Webb, Dave Syer, Josh Long, Stéphane Nicoll, Rob Winch, Andy Wilkinson, Marcel Overdijk, Christian Dupuis, Sébastien Deleuze, and Michael Simons. 2017. *Spring Boot Reference Guide 1.5.4.RELEASE*. https://docs.spring.io/spring-boot/docs/1.5.4.RELEASE/reference/pdf/spring-boot-reference.pdf.

[34] Catherine S Weir, Gary Douglas, Martin Carruthers, and Mervyn Jack. 2009. User perceptions of security, convenience and usability for ebanking authentication tokens. *Computers & Security* 28, 1 (2009), 47–62.

[35] Thomas Y.C. Woo and Simon S. Lam. 1993. A Semantic Model for Authentication Protocols. In *Proceedings of IEEE Symposium on Research in Security and Privacy*. 178–194.

[36] Margaret Woodward. 2013. Air Force Instruction 91-104. https://fas.org/irp/doddir/usaf/afi91-104.pdf.