On the Security of In-band Verification in End-to-End Encrypted Video Calls

Daniel Jones*
Independent Researcher, UK
daniel@wilshirejones.com

Melissa Chase
Microsoft Research, USA
melissac@microsoft.com

Esha Ghosh
Microsoft Research, USA
esha.ghosh@microsoft.com

Kim Laine
Microsoft Research, USA
kim.laine@microsoft.com

Abstract

Video conferencing software, including Zoom, Microsoft Teams, and Cisco Webex, use human-driven key verification ceremonies to protect end-to-end encrypted meetings against a potentially malicious service provider. The client software shows each participant a code that they must compare; if the codes match, the call is considered secure. Prior security analyses assumed authenticated out-of-band channels for the comparison, but this is generally unrealistic. The codes are short-lived, so the "in-band" channel being verified is, itself, the most natural one to use. We seek to understand the implications of this common practice, asking whether it can be secure and under what conditions.

To this end, we formalize the notion of a *Human-to-Human Group Key Agreement* protocol, modeling an authenticated group key exchange between *people*, rather than their cryptographic keys. We identify that the security of these protocols relies on the pre-existing capacity of people to consistently recognize one another, avoiding any global identification scheme or trusted external infrastructure.

We present a construction, prove it secure in our model, derive concrete bounds, and discuss non-examples demonstrating the definition's subtlety. Our results highlight the approach's usability issues and reliance on unforgeability of human-authenticated video streams—an assumption additionally challenged by recent advances in deepfakes.

1 Introduction

Informally, *end-to-end encryption* (E2EE) is a security paradigm where two or more endpoints interact over an untrusted network with (at least) data confidentiality and integrity [31, 56]. The adversary can intercept, manipulate, and create arbitrary messages, but not break these guarantees.

Setting up an E2EE channel requires the participants' public "identity" keys, which the service provider distributes in nearly all deployments. Participants must verify key distribution is done correctly in a process called *key verification* [25, 31], as otherwise an adversarial service provider can distribute fake keys and imperceptibly intercept all traffic with a man-in-the-middle (MitM) attack.

Approaches to key verification. A common method that avoids trusted third-parties is to verify key distribution over an existing ***out-of-band authenticated channel***. For example, Signal [52], Discord [21], and some deployments that use the MLS protocol [6] compute a security code: a (possibly truncated) hash of the public keys or some part of a key exchange transcript. The idea is to use this channel to check that everyone's (trusted) client computed the same code and conclude the keys must have been distributed correctly [46]. The problem is that such authenticated out-of-band channels (including meeting in person to scan QR codes) are not easily available.Some deployments use pairwise codes, requiring a quadratic number of comparisons to verify a group.

In-band key verification. In larger meetings or enterprise settings, participants may not know each other, much less share an out-of-band channel. It has thus become common to apply similar techniques ***in-band***: users compare security codes by reading them aloud over the encrypted call. This is something of an industry standard for enterprise E2EE meeting services.[1] Some of the largest providers like Zoom [10, 61], Microsoft Teams [42], and Cisco Webex [20] support in-band verification.[2]

Surprisingly, there has been very little work analyzing this type of protocol, or even describing its assumptions and intended security goals. This is our focus: comparing verification codes in-band without relying on existing authenticated out-of-band channels, trusted third-party infrastructure, or

*Work done in part during an internship at Microsoft Research.

[1] Consumer E2EE platforms such as WhatsApp, Apple iMessage, and Signal allow calls only over an established E2EE messaging channel, whose key verification is out-of-band. This may explain their lower adoption of the in-band method than enterprise platforms, which tend to prioritize meetings.

[2] Only Zoom's wording is entirely unambiguous that the code is verified in-band, but we interpret all three as intending this.

global user identifiers. We call this the *in-band security code method*, following the terminology in [61].

We formalize and analyze this widely deployed approach, making explicit the assumptions those who already use such ceremonies *implicitly* rely upon. Our objective is not to show in-band security codes (including the protocol we present) are an optimal way to verify public keys; indeed, our results highlight a number of usability issues, as well as a reliance on the unforgeability of human-authenticated video streams—an assumption challenged by recent advances in deepfakes. If users do have well-known identifiers, verification based on key transparency may be preferable; we return to this in Section 5.

1.1 Other Approaches

We briefly highlight the most relevant alternatives; Appendix H discusses related work—including the out-of-band model, SAS, and ZRTP—in more detail.

SAS and ZRTP. In-band key verification has some resemblance to Short Authenticated Strings (SAS) [35, 46, 57] and similar works [33, 34, 44], which assume authenticated out-of-band channels for comparing short strings. The setting in these papers differs too much from ours for their results to translate meaningfully, as we see at the end of Section 1.4. ZRTP [15] applies an SAS protocol directly to in-band verification.

Key transparency. Key transparency [19, 28, 37–39, 41, 55] aims to partially automate key verification: rather than having to actively participate, users monitor event notifications and act only if something unexpected occurs. This promising direction has recently seen deployments by WhatsApp [36], Apple iMessage [4], and Proton Mail [27]—all of which still include out-of-band verification as a fallback.

As E2EE meeting providers consider moving from security codes to key transparency, understanding the current approach is necessary to assess the trade-offs taken in such a switch. This paper aims to provide a firm basis for comparison with key transparency and other future key verification paradigms.

1.2 The In-Band Security Code Method

In-band security code verification has each client (software) compute a security code summarizing the list of participating public keys or parts of its key exchange transcript. The client displays this code to its user, who is typically instructed to "verify that everyone sees the same code" by reading it out loud.[3]

The intuition is that if Alice sees Bob read the same code her client displays, she knows his client displays the same code to him. If all clients in the meeting display matching codes, and the code correctly summarizes the security-critical parts of the key exchange, then they all share essentially the same view of it. If the key exchange succeeds, the shared key is known only to those clients.

This has inherent disadvantages. First, reading and verifying strings roughly 20–40 digits long is cumbersome. Second, it assumes that convincing fake video of another user reading a specific code is hard to produce; as deepfake technology improves without matching advances in detection, this assumption becomes less realistic. Finally, because it relies on users reading and watching one another read these codes, it is unclear how it could be made accessible to users with disabilities that prevent them from reading aloud, or from seeing or hearing another user read the code.

However, in-band security codes also have interesting advantages. No additional infrastructure or trusted third party is needed, and security codes can be implemented using only ephemeral public keys with no long-term key storage beyond the meeting. This naturally provides a degree of forward secrecy and post-compromise security between sessions. The security code method does not require users to have external relationships with—or know anything about—the other participants, beyond recognizing them as the people who read the code. In particular, it works even when users share no other communication channel and when different users know the same person by different identities. Interestingly, it can provide Alice with a security guarantee in an asymmetric scenario, where Alice recognizes and trusts Bob, but not vice versa. This is valuable, e.g., in whistleblower scenarios, where one endpoint is a publicly known journalist and the other wants to keep their identity hidden.

The need for analysis. At first glance, the in-band security code method might seem straightforward enough to take for granted, making detailed analysis seem unnecessary. We argue that this is very much not the case.

First, the description above is vague and, as we discuss in Section 1.4 below, the details of the implementation matter greatly. For example, it is unclear from the above—or, in fact, from the documentation for some of the widely used products relying on this method—how exactly users are expected to "verify that they see the same code." Is it enough that one user reads it and the others proceed as long as their codes match? Must the users respond somehow to confirm a match? Should all users read the code instead? Analyzing these options, and other subtle details of the protocol, requires a clearly specified security definition, in particular one that captures the human component of the protocol.

Second, it is unclear exactly what assumptions this approach relies on. An adversary who can instantaneously simulate arbitrary videos of users can clearly break it, but what about one that only records and replays video? Must the user keep their video on for the entire call, or only for the portion

[3] An example of this phrasing can be found in the Cisco Webex documentation [20, Page 9].

where the security codes are compared? What we need is a concrete model describing the assumptions underlying the in-band security code approach.

Finally, what security do we get under these assumptions, and how does it compare to traditional key exchange with a PKI? The meeting setting is fairly different: rather than knowing the participants of a key exchange ahead of time, current E2EE meeting platforms allow ad-hoc meetings and calls, where a user can receive a call, decide to answer, and only then find out who is on it; our security model must be meaningful in this setting as well. Our model must also capture the advantages discussed above: constructions cannot require globally consistent identifiers and should provide a security guarantee even if there is asymmetric trust among users.

1.3 Our Contributions

A new protocol definition. We define Human-to-Human Group Key Agreement—human-to-human because it models a secure connection between the actual users, not just their devices: Alice learns she is in a meeting with Bob, not merely with a device holding a particular public key. Our definition enables realistic security statements about users communicating over modern E2EE meeting platforms; it allows for an adversary that can initiate meetings and invite users to join, so that users learn who is in the meeting only once they have joined. It provides concrete guarantees regarding "learning who is in the meeting," even where a user may be known to others by different identities, and it provides meaningful security for a participant who alone can recognize (and trust) all the others.

An idealized "human authentication" functionality. We present an idealized model capturing the assumptions inherent to the in-band security code method. Our human authentication ideal functionality allows users to "authenticate" messages and, upon receiving one, identify both its producer and its content. Since we must realistically rely solely on people's ability to recognize one another, without external infrastructure to assert a global identification scheme, we build our notion of authentication from *identity consistency*: a recipient is assumed *only* to be able to consistently recognize a person they have encountered before, and never to attribute an artifact to the wrong person. Without globally agreed identifiers there is no "real Bob", but there is a person that Alice will consistently identify as Bob, which suffices for her to expect secrecy with him. Intuitively, proving a protocol secure with the real human process replaced by this ideal functionality [8, 16] leaves only those attacks that break the assumptions of this functionality—by generating a deepfake of a user reading a message, for example.

A new construction. Finally, we present a new protocol and prove it a secure Human-to-Human Group Key Agreement in this idealized model.

We analyze our own protocol rather than a deployed one, partly because deployed protocol details are unclear or unavailable: Microsoft Teams and Cisco Webex provide few details, and while Zoom analyzes its cryptographic components well, its application UI and documentation contradict the whitepaper [10] in their description of the human component.[4] Zoom also assumes a clearly specified "leader" participant, complicating both the UI and the analysis; for simplicity, we focus on protocols where the UI does not (need to) identify such a "leader" user.

Avoiding deployed systems also lets us present a simplified protocol containing only what our security goals require, easing the analysis and future implementations. It can be proved secure according to our definition with codes of length comparable to current deployments. It decomposes into a cryptographic group key exchange component and a human authentication-based component verifying that a group of users share the same security code, letting us analyze the two separately to better understand the security of the composition.

Security goals and threat model. We consider an attacker who has compromised the network, remote service provider infrastructure, and a subset of users. Beyond the human authentication functionality discussed above, we assume no trusted PKI, out-of-band channel, or global identification scheme.

Say a user *accepts* if they complete the key exchange and all other participants they believe to be in the call are honest. Our goal is to guarantee (a) *key indistinguishability*: if a user accepts, the resulting key is hidden from the attacker; (b) *authentication*: if a user completes the protocol, every *honest* participant they believe to be in the call takes part in the same key exchange; and (c) *match security*: if every user in a key exchange accepts, they all agree on the key and participant list. These properties must hold even when the attacker schedules calls arbitrarily and users do not know the other participants ahead of time.

1.4 Details Matter

At a high level, our protocol works as follows. The service provider sends each party the list of all other parties' public keys. One party chooses a random symmetric key and encrypts it under each of these keys; each party decrypts the appropriate ciphertext to retrieve it. Finally, each party hashes the public keys it was given, and all verify that their hashes match by reading them aloud over the call. This seems

[4] The whitepaper [10, Section 7.7] requires every participant to read the security code, the leader first, whereas the user-facing documentation [61] and application UI ask that the meeting host should read the security code aloud and the meeting participants verify that their codes match. We have informed Zoom of this issue in July 2026.

straightforward, but getting from this high-level intuition to a detailed protocol is surprisingly subtle: small changes can result in a completely insecure scheme. We illustrate with five attacks.

Attack 1. Consider the following instantiation. Each party generates two long-term key pairs—one for a digital signature scheme, one for public key encryption—and stores them with the server. When a meeting starts, the server distributes the associated list of public keys and designates one user as the *leader*; the leader generates a fresh symmetric key and encrypt-then-signs it for each non-leader, who verify-then-decrypt to receive it. The security code is the hash of all public keys in lexicographic order.

Unfortunately, a replay attack lets the adversary insert itself into the meeting undetected. When Alice's and Bob's clients initiate the meeting, the adversary initiates two different key exchanges. The first includes Alice, Bob, and an adversarial session, designating Alice as the leader; Alice's client chooses a fresh symmetric key and encrypts-then-signs it to both Bob and the adversary. The adversary immediately aborts the meeting (the server can claim, e.g., "network failure"), storing the key and the signed ciphertext for Bob, then repeats the process with Bob as the leader. Finally, it sets up an honest key exchange between Alice and Bob, but tells Alice that Bob is the leader and Bob that Alice is the leader, and replays the saved key distribution ciphertexts to Bob (from the first session) and Alice (from the second). Knowing both keys, it can insert itself into the meeting, and security code verification passes because Alice and Bob have matching participant lists.

One may try to fix this by making the leader's identity explicit in the security code, as Zoom does; in fact, its security codes are derived only from the leader's information [10, Section 7.7].

Attack 2. Encoding the leader's identity into the security code mitigates Attack 1, but the protocol remains vulnerable: the adversary can impersonate Alice to other meeting participants as follows. It initiates a meeting between Alice, Bob, Carol, and an adversarial session with Alice as the leader; as above, it stores the signed ciphertexts intended for Bob and Carol, then terminates the meeting. Next, it initiates a meeting between Alice, Bob, and Carol, again with Alice as the leader, and distributes the previous ciphertexts. Everyone obtains the same security code and begins the encrypted portion of the meeting. The adversary now tells Alice the call has failed and impersonates her to Bob and Carol for the remainder, e.g., by replaying old recorded video or turning the camera off.

This attack is one-sided in that the adversary must impersonate Alice and cannot MitM the meeting, but it is still a concerning problem that must be prevented. The key issue is that the key distribution messages carry too little information about the current meeting. Zoom solves this by having the non-leader members choose new ephemeral keys for each meeting and adding a regular heartbeat message, which encodes a history of all joining/leaving users and public keys. We choose a simpler approach: all participants, including the leader, choose a new key pair for each run of the protocol. Each party checks that its own fresh key appears in the list it is given and computes its security code from that list, so a malicious server cannot reuse key distribution messages from earlier runs.

To fix these problems, we only use ephemeral keys in our construction. The order of the keys indicates the leader: the first is the leader's signature verification key and the rest are in lexicographic order. The security code is computed from the list sorted this way, so verification fails unless all clients agree on the session and have the same view of the leader.

Attack 3. Ephemeral keys address the problem enabling Attack 2, but another weakness in the key exchange protocol may again let an adversary insert itself into a meeting undetected. The problem is that the security code must be short enough to be practical to read. Since the protocol described thus far lets the adversary see all honest public keys before choosing its own, it can mount an offline birthday attack to find a security code collision: it starts executing two key exchange protocols with Alice and Bob separately and, upon seeing their public keys, chooses two public keys for itself that make the security codes match. Forwarding all video from Alice to Bob and vice versa, security code verification passes between Alice and Bob, yet the adversary intercepts all communication.

We address this with a commit-and-reveal technique. During session initialization each client commits to a random nonce by hashing it, and distributes the commitment along with its public key. With the request to join a key exchange protocol, clients expect to receive the others' commitments and public keys, and respond with their nonces. To complete the key exchange, each client is given a list of nonces, verifies that they hash to the commitment values, and computes the security code from the keys and the nonces. This protects against offline birthday attacks, because the adversary must proceed all the way to the last stage of the protocol to see whether the codes match, thus requiring interactivity. A similar problem and solution are described for ZRTP in [15, Section 4.4.1.1].

Attack 4. Attacks 1–3 concerned the cryptographic part of the key agreement protocol. With these mitigated, we turn to the human ceremony verifying that all users on the call have the same security code. Consider a naive verification ceremony where one participant—the *speaker*—communicates the security code to the group and waits for a confirmation from each of the other members, the *verifiers*. Notably, the verifiers confirm with a simple acknowledgment; they do not repeat the security code. The speaker accepts the verification as valid once every verifier has confirmed, and each verifier accepts the session only after hearing the correct security code from the speaker and confirmations from all other verifiers.

This may work in person, but is not necessarily secure online. In person, participants can rely on the group having a consistent view of who is present and of what those participants say.[5] Online, verifiers cannot know which security code the others are confirming, or even that they are responding to the same person, which can let an adversary insert itself into a meeting undetected.

Consider a simple attack against Alice and Bob. First, the adversary needs each of them to participate as a verifier in some call that also contains adversarially controlled parties, from which it can record each party's acknowledgement (e.g., thumbs up). It then creates a call between Alice and a fake session representing Bob, and one between Bob and a fake session representing Alice, starting a verification ceremony for both sessions with the honest party as speaker. Since the speaker expects only a simple acknowledgment, it suffices for Mallory to play the prerecorded acknowledgment to each honest party, showing *the other party* seemingly confirming the code. Both parties then believe themselves to be in a secure call with one another, when in fact both are communicating with the adversary. The adversary must then either claim each honest party has turned off their video, or somehow splice the prerecorded acknowledgment onto the current video stream; from that point on it relays both video feeds between the authentic parties, thus having inserted itself in the middle. A similar attack against ZRTP with an in-band one-way verification ceremony is discussed in [47, Section 4.3.3]; ZRTP was originally used for audio calls, which makes it particularly easy to splice a prerecorded acknowledgment onto the current stream.

One could try to fix this by requiring participants' video to be continuously on, assuming that such video is hard to generate. This is already dubious—in a large meeting a prerecorded video of Alice saying nothing might suffice—but the next attack works even with this attempted mitigation.

Attack 5. Meeting participants often lack prior knowledge of each others' identities, or the knowledge is only one-way (e.g., employees recognize their CEO). An adversary can exploit this to insert itself into a meeting undetected, as follows. Recall that we want our protocol to provide security for Alice as long as she can recognize and trust the other participant(s) in the call, even if they do not recognize her (the whistleblower scenario). The adversary schedules two underlying sessions for this meeting: the first between (adversary-controlled) Mallory and Bob, the second between Alice and Mallory, but with Mallory's video replaced by Bob's video from the first meeting. The two calls have differing security codes, so we would expect the verification ceremony to detect this. Suppose Alice is chosen as the speaker. Rather than repeating the security code voiced by Alice, Mallory simply reads out the security code shared with Bob; when Bob confirms it, the forwarded video appears to Alice as a confirmation of the code she read. From Alice's perspective, she has successfully completed in-band verification with Bob. This attack relies on Bob not having prior knowledge of Alice's or Mallory's identities.

Our solution to both Attacks 4 and 5 is simply to require every participant to speak the security code. This is burdensome, but these generic attacks show that any in-band protocol where only one participant reads the code is vulnerable, as previously documented for ZRTP [47, Section 4.3.3]. This observation shows a core difference between our setting and that considered by the SAS work [33,34,44,57]: [43, Section 4.1] specifically argues that a variant of one-way verification is sufficient in their setting (see Appendix H, *On the (in)security of one-way ceremonies*). Whether a given deployment is vulnerable to Attack 3 comes down to implementation details that are generally not public. Attacks 1 and 2 target a naive baseline and serve only to motivate our construction.

2 Human Authentication

Our goal is to model the ability of people to "authenticate" each other through video artifacts: roughly, the process where one person communicates a message in a way that another person can interpret the message, recognize tampering, and recognize that it was created by a unique other human. The process must blend into synchronous communication between these and possibly other participants.

W develop an *idealized* human authentication functionality $\mathcal{H}$, responsible for generating new identities, managing them consistently, and simulating the ability for people to correctly and consistently authenticate one another. This idealized functionality [8,16] is not intended to be instantiated by a real-world protocol, but rather models the assumptions about real-world human interaction that are necessary to establish security.[6]

A proof relative to $\mathcal{H}$ rules out any attacks that do not exploit the ability of people to, in a precise sense, consistently authenticate messages (encoded as bit strings) received from each other; $\mathcal{H}$ thus isolates and names these assumptions, revealing exactly where the industry standard security code method might fail. Deepfakes and identical twins are examples of threats that would break these assumptions. Voice-only channels, specifically, are not a viable option due to their high susceptibility to mimicry [2, 45, 51, 59]; real deployments in video conferencing software use live video of a person reading the code.

[5] In the terminology of group key exchange, in-person groups can rely on *transcript consistency* and *participant consistency* [49,56].

[6] In fact, $\mathcal{H}$ makes normative assumptions about how people communicate, many of which do not universally apply, even though real-world deployments rely on them (see Section 5).

2.1 Identifiers and Authenticated Messages

Consider a conference call where Alice recognizes her colleague Bob by face and voice, but there is no PKI, no pre-shared keys, and no out-of-band channel binding Bob's video feed to a global name. Alice's identification of Bob is purely local—she knows him, but Carol (who has never met Bob) does not. Another participant, David, might know Bob as a man named Robert instead. Thus, we cannot rely on global identifiers that everyone would agree on; our system must be secure assuming participants recognize each other only using local identifiers. That said, the challenger in our security games needs to link local identifiers to the person behind them unambiguously, clearly necessitating global identifiers. However, these are merely an artifact of the proof technique and do not (need to) exist to the protocol participants. Concrete constructions must only use local identifiers.[7]

Concretely, consider a protocol potentially used by n_p people. A global indexing scheme $i \in [n_p]$ identifies each person so that the participants in any protocol execution form a subset of $[n_p]$. In practice, each party maintains a local indexing scheme, which allows them to recognize others they have interacted with before. Locally, each party assigns the index 0 to themselves and indices from 1 onward to each new party they interact with. With this scheme, if Bob has a meeting with Alice, we expect that (a) Bob recognizes her as the same person in later meetings, and (b) Bob does not misidentify Claire as Alice.

To model human-authenticated messages, each party $i \in [n_p]$ is given access to a pair of algorithms $(\mathsf{Auth}_i, \mathsf{Ident}_i)$:

$\mathsf{Auth}_i(m) \to \hat{m}$ captures party i generating a human-authenticated message $\hat{m}$ with content m. Concretely, in our protocol $\hat{m}$ represents a video of person i reading a security code m.

$\mathsf{Ident}_i(\hat{m}) \to (\hat{j}, m)$ captures party i interpreting a human-authenticated message $\hat{m}$ to determine its content m and a local identifier $\hat{j}$ of the party that generated it.

Letters $i, j, \ldots$ indicate parties in the global indexing scheme, whereas $\hat{i}, \hat{j}, \ldots$ indicate parties in the local indexing scheme of a party.For example, $\mathsf{Ident}_i(\hat{m})$ outputs $\hat{j}$, corresponding to party j in party i's local index.

We represent communication in distinct chunks (messages), themselves inseparable but possibly reordered or manipulated by the adversary. Thus, in our modeling, we can talk of a party sending a (human-authenticated or not) message to another party (through a router). The intended receiver may or may not receive the message that was sent.

[7] Looking forward, the formalism presented in Section 3 ensures *not* to expose the global identifiers to the construction.

2.2 Encodings

To be useful, human-authenticated messages must be transferable from one party to another, so we must define how such a message is *encoded* into (and *decoded* from) a redistributable bit string. We let the adversary pick the two algorithms, Encode and Decode, for this purpose. Encode is probabilistic; it takes as input a party identifier $i \in [n_p]$ and a message $m \in \mathcal{M}$, which it transforms into a bit encoding $\hat{m} \in \{0,1\}^*$. Decode is deterministic; it takes a bit encoding and outputs the encoded message and the party identified as its originator.

Not every choice of Encode and Decode is appropriate to use. We capture our requirements on these algorithms through the ideal functionality in the next section, which sets an error flag when they do not behave as expected (once the flag is set, the functionality refuses to answer further queries). For example, we require that a message encoded with Encode is correctly recovered by Decode. Appendix G provides example instantiations demonstrating that our requirements are satisfiable.

2.3 Ideal Functionality

Our human authentication ideal functionality $\mathcal{H}$ is presented in Figure 1. It consists of the oracles $\mathcal{H}.\mathsf{Init}$, $\mathcal{H}.\mathsf{Gen}$, $\mathcal{H}.\mathsf{Auth}_i$, $\mathcal{H}.\mathsf{Ident}_i$, and $\mathcal{H}.\mathsf{Mapback}$. This is roughly inspired by the signature functionality in [17], in particular the variant on hiding the signed message that they mention in their discussion after presenting the functionality.

The functionality is initialized by calling $\mathcal{H}.\mathsf{Init}(\mathsf{Encode}, \mathsf{Decode})$. Looking ahead, in the security experiments (Figure 2), the challenger calls $\mathcal{H}.\mathsf{Init}$ using Encode and Decode obtained from the adversary, to capture a worst case choice of encoding scheme.

After checking that no error flags are set, $\mathcal{H}.\mathsf{Gen}()$ creates a new human identity, returning its identifier i in the global indexing scheme and pointers to two oracles, $\mathcal{H}.\mathsf{Auth}_i$ and $\mathcal{H}.\mathsf{Ident}_i$. The caller uses these to create human-authenticated messages from party i and to verify human-authenticated messages from other parties from the perspective of party i, respectively.

Looking ahead, both the challenger and the adversary have access to $\mathcal{H}.\mathsf{Gen}$ in our H2H-GKA security experiments. The challenger uses it to generate *honest* human identities, while the adversary may use it to generate adversarial ones. Only the challenger can act on behalf of honest human identities and delegates this ability to honest protocol sessions, while only the adversary can act on behalf of adversarial ones.

The $\mathcal{H}.\mathsf{Auth}_i$ oracle creates a human-authenticated message $\hat{m}$ on behalf of a party i for a message m. After checking that no error flags are set, it uses Encode to generate the bit string $\hat{m}$ and runs Decode on $\hat{m}$ to test whether Decode correctly reverses the encoding; otherwise, it sets

the *InconsistentDecoding*? error flag. Next, it checks whether $\hat{m}$ is already in a set of rejected messages, in which case it sets the *CheatWithPrediction*? error flag; this indicates a temporal inconsistency, where a message $\hat{m}$ was first rejected and later accepted.[8] Finally, it stores the tuple $(i, m, \hat{m})$ in a table AM to keep track of legitimate human-authenticated messages.

The $\mathcal{H}.\mathsf{Ident}_i$ oracle is used to (attempt to) decode a bit string $\hat{m}$ into the producer's party identifier from the perspective of party i along with the encoded message m. After checking that no error flags are set, the oracle runs Decode to determine the encoded message m and the party j that generated the message. Next, it checks that the table AM contains a tuple $(j, m, \hat{m}')$, indicating that party j indeed authenticated the message m.[9] If Decode produces m and j for which there is no corresponding entry in AM, we indicate this (forgery) as a verification failure by returning $\perp, \perp$, and we add $\hat{m}$ to the rejected message set RM to prevent temporal inconsistencies, as discussed above. Otherwise, we return the local party identifier of the producer (from the perspective of party i) along with the message m.

Finally, the $\mathcal{H}.\mathsf{Mapback}$ oracle maps a party-local identifier (from the perspective of a given party) back to a global identifier: $\mathcal{H}.\mathsf{Mapback}(j, \hat{k})$ returns an identifier k in the global index such that $\hat{k}$ is party j's corresponding local identifier for k. If the second parameter is a list (resp., a set), the oracle applies to each element in turn, returning a list (resp., a set) of global identifiers. $\mathcal{H}.\mathsf{Mapback}$ allows challengers to track the different parties in security experiments; it is not available to protocol sessions, as the global index does not exist in the real world.

2.4 Properties of $\mathcal{H}$

In Appendix E we derive four natural properties from Figure 1. Roughly, they capture that any recipient consistently and correctly recognizes the content of a human-authenticated message and consistently and correctly identifies the speaker. In particular, the "authentication" is unforgeable and different recipients cannot interpret the same human-authenticated message in different ways. While these properties are not directly used in the rest of this work, they add intuition as to why Figure 1 constitutes a meaningful definition.

3 Human-to-Human Group Key Agreement

In this section, we define Human-to-Human Group Key Agreement (H2H-GKA) as an interactive protocol and ceremony that allows a group of parties (people running client software) to establish a shared secret over an insecure network, *without* relying on any trusted third-party infrastructure. To make this possible, it additionally models the ability of people to send human-authenticated messages over the same network the key exchange is performed over.

Processes between client applications are defined as *protocols* and between humans as *ceremonies* ([26]). We describe a combined protocol and ceremony, considering a person and the client they control as a single entity (more details in Appendix A.2). Thus, we model communication between client devices, but also how humans work alongside their devices to complete the protocol.[10]

Our security model is centered around a group key exchange formalism [7, 9, 11–13, 18, 40], which follows closely from the model proposed in a recent SoK paper by Poettering et al. [48]. Crucially, and unlike related work in the out-of-band setting [34, 35, 43, 44, 46, 50, 57], our model assumes neither distinct authenticated channels nor a consistent global identification scheme for the people involved. This lets us capture sessions that need not be initialized with a trusted command describing the group, such as a pre-existing identifier or a member list. Instead, all initialization messages are sent over the untrusted channel, just as is often done in practice.

Further, security requires that the key remain secret even if *another* honest participant has computed a member list with adversarial parties, as long as the session in question has accepted with a member list that solely contains honest members. This means that, as long as Alice trusts all those she sees in the meeting, the resulting key is secure.

Syntax. At a high level, the protocol starts by initializing some number of sessions, each provided with no initial input to guide it. Together, these sessions execute a protocol, where they jointly establish a shared secret key and discover the "human identities" of each other. At the end, each session either accepts the key, key context, and group membership (as a set of party-local identifiers) in their private state, or rejects the session. The key context can be thought of as a handle that separates one key exchange from another; for example, if Alice and Bob have multiple distinct meetings between them, the key context would identify which exact meeting they are currently a part of. A concrete instantiation of this handle can be a concatenation of the participants' long term and meeting-specific ephemeral keys.

Concretely, human-to-human group key agreement is an interactive protocol and ceremony. We define its behavior, primarily, through a state machine, $\Pi^{\mathcal{H}.\mathsf{Auth}_i, \mathcal{H}.\mathsf{Ident}_i}_{\mathsf{H2H\text{-}GKA}}(st, m) \to (st', m')$, which describes the behavior of each party i and has oracle access to that party's human authentication interfaces. A session is initialized by calling the state machine with an

[8] For example, if $\mathcal{H}.\mathsf{Auth}_i(m)$ represents a video recording in which a party i reads a message m, then *CheatWithPrediction*? indicates that the adversary could predict the video before it was recorded.

[9] The original encoding $\hat{m}'$ need not match the encoded message $\hat{m}$ passed to $\mathcal{H}.\mathsf{Ident}_i$; this captures the fact that, for example, the video artifact may be resized or compressed without changing how it is interpreted.

[10] This formalism does not capture the communication channels *between* a user and their device. See [54] for an example of work that does capture such interactions.

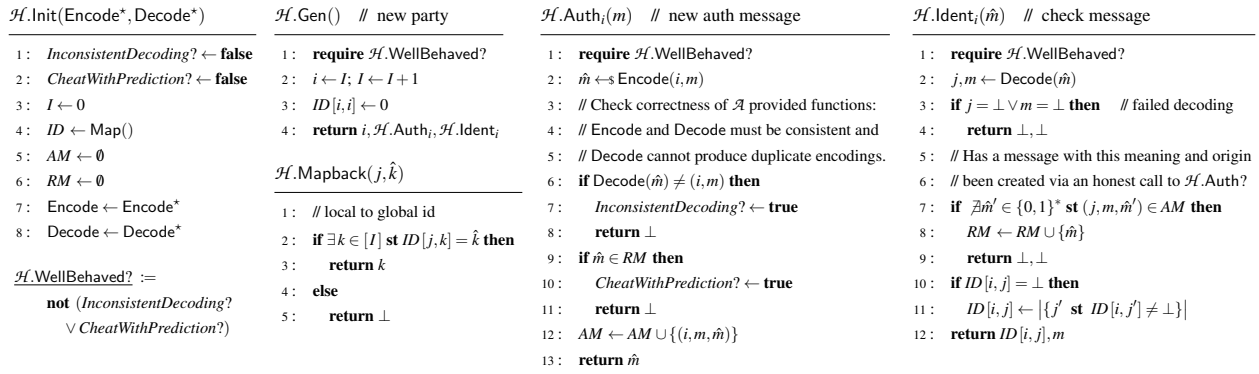

Figure 1: An idealized implementation of the human authentication functionality, $\mathcal{H}$.

empty initial state and message. The protocol proceeds with each session processing a series of requests, each of which it may choose to respond to, and each session is expected to learn (through these interactions) the identities of the group members in addition to the group key. We require the state machine's state, st, to include the following public and private fields: (1) $st.x$ storing the current status (public), (2) $st.kid$ storing the key context (public), (3) $st.mem$ storing the group membership list (public), and (4) $st.k$ storing the shared key (private).

In the end, $st.x$ must be set to either `accept` or `reject`, denoting that this session succeeded or failed in the protocol execution, respectively. Success ($st.x = \texttt{accept}$) indicates that, from the perspective of this party, the group $st.mem$ agreed on a shared key $st.k$ with key context $st.kid$; here $st.mem$ is a set of party-local identifiers for the group members.

Security. Intuitively, we expect a secure H2H-GKA protocol execution to guarantee that if a group of parties succeeded in executing the protocol, then they must all agree on the membership list of the group, they must have computed the same shared key and context, and the key must be known only by the members of the group. Otherwise, at least one of the sessions must have rejected. Formally, we require three security notions: *match security*, *key indistinguishability*, and *authentication*; the corresponding security experiments are presented in Figure 2. These guarantees must hold even when all communication occurs over an untrusted network, where the adversary can manipulate or route messages arbitrarily. Appendix B details our security experiment. Throughout the three definitions below, Λ are the parameters used to instantiate Π, n_p is the number of parties, and n_s is the maximum number of sessions.

We define two sessions to be *partnered*, or *matching*, if they arrive at the same key context, *kid*; conceptually, the protocol believes partnered sessions to have participated in the same protocol execution. Our security experiments use this to determine whether the corresponding keys are expected to be related.[11]

Since we allow a protocol to define its own partnering, we must ensure that this partnering is valid. *Match security* does just that: it captures the consistency guarantees required between groups of sessions that the protocol believes are part of the same protocol execution. This is seen most clearly by considering a passive adversary that faithfully forwards messages between sessions without any modification: we would expect any group of matching sessions to have computed the same secret key, agree on the members of the group, and to have done so accurately (every matching session should be accounted for in the agreed upon member list).

To allow for active adversaries, we only expect sessions that have accepted to have a consistent view, as accepting indicates a successful completion of the protocol. However, we still require that the consistent view reflect the whole group, including parties that have not (yet) accepted.[12] We also cannot require anything from groups of sessions where any session includes an adversarially controlled party in its member list, because the challenger cannot track the (possibly) matching sessions controlled by the adversary. This mirrors the real-world ability of a misbehaving party to break the consistency guarantees of the protocol.

Definition 1 (Match security of H2H-GKA)**.** The advantage of an adversary, $\mathcal{A}$, against the match security, as defined by Match in Figure 2, of a human-to-human group key agreement protocol, Π, is

$$\mathsf{Adv}^{\Pi,\Lambda,n_p,n_s}_{\mathsf{Match}}(\mathcal{A}) := \Pr\left[\mathsf{Match}^{\Pi,\Lambda,n_p,n_s}_{\mathsf{H2H\text{-}GKA}}(\mathcal{A}) \mapsto 1\right].$$

The *authentication* property is considered from the perspective of a single session that has accepted, without any external knowledge of the status or honesty of its partners.

[11] This can be seen in our use of the sets *CH* and *WK* in Figure 2, and how these sets are used to determine which other sessions can be challenged.

[12] For example, accepting sessions must correctly calculate the group membership to include those parties that computed a matching key context but have not accepted.

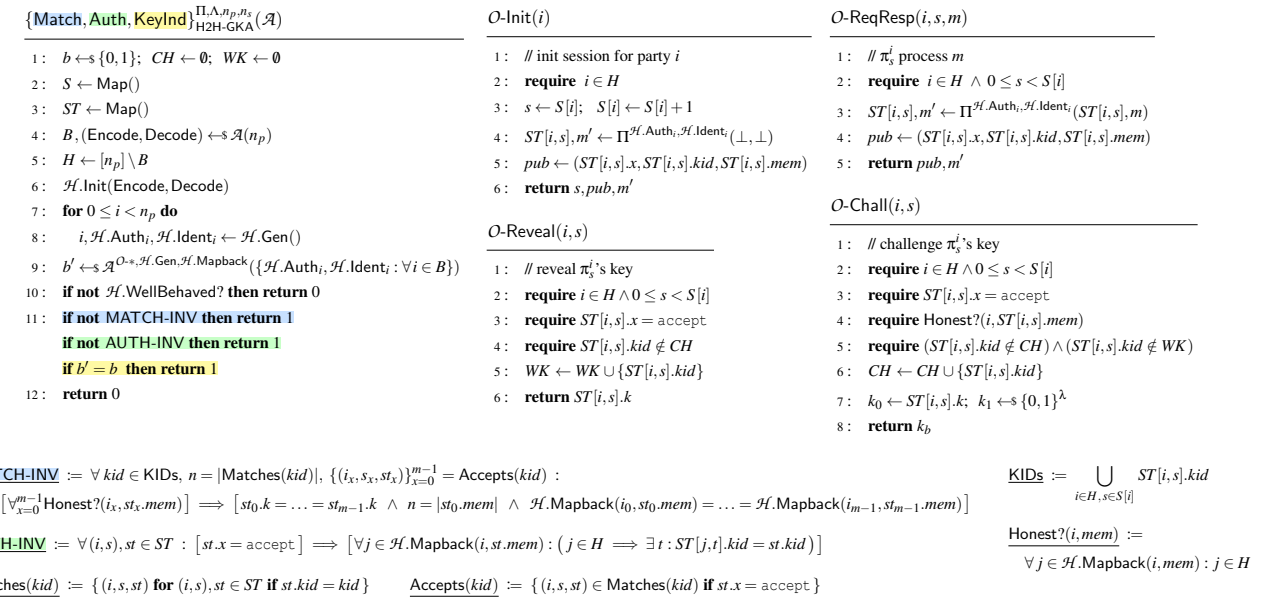

Figure 2: Security experiments defining match security, authentication and key indistinguishability for Human-to-Human Group Key Agreement with respect to a human authentication functionality, $\mathcal{H}$.

We simply require that whenever a session accepts with a particular membership list *mem*, there exists a matching session for every honest party in that list.

Definition 2 (Authentication of H2H-GKA)**.** The advantage of an adversary, $\mathcal{A}$, against the authentication, as defined by Auth in Figure 2, of a human-to-human group key agreement protocol, Π, is

$$\mathsf{Adv}^{\Pi,\Lambda,n_p,n_s}_{\mathsf{Auth}}(\mathcal{A}) := \Pr\left[\mathsf{Auth}^{\Pi,\Lambda,n_p,n_s}_{\text{H2H-GKA}}(\mathcal{A}) \mapsto 1\right].$$

We capture the secrecy of keys through *key indistinguishability*, where the adversary is challenged to distinguish real keys, generated by an honest execution of the protocol, from uniform random bits. As explained above, challenges can be submitted via the O-Chall oracle for any session satisfying the following prerequisites. First, it must have accepted (failed or incomplete sessions are not expected to generate secure keys). Second, its membership list may only contain the identities of honest parties; if an adversarially controlled party is an *expected* member of the group, we cannot expect the key to be secure. This captures a requirement on the protocol to accurately keep track of human identities. Third, the adversary must not have been given direct access to the real key by querying the O-Reveal oracle against this session or a session partnered with it.[13] Finally, none of the challenged session's partner sessions may have previously been challenged.

Definition 3 (Key indistinguishability of H2H-GKA)**.** The advantage of an adversary, $\mathcal{A}$, against the key indistinguishability, as defined by KeyInd in Figure 2, of a human-to-human group key agreement protocol, Π, is

$$\mathsf{Adv}^{\Pi,\Lambda,n_p,n_s}_{\mathsf{KeyInd}}(\mathcal{A}) := \Pr\left[\mathsf{KeyInd}^{\Pi,\Lambda,n_p,n_s}_{\text{H2H-GKA}}(\mathcal{A}) \mapsto 1\right] - \frac{1}{2}.$$

Does this model capture our attacks? We consider each attack introduced in Section 1.4 in an instance of the H2H-GKA experiments. We assume (as our construction does) that the security code is used as the key context *kid*.

Attack 1 runs three key exchanges. The adversary aborts the first two; note that it cannot challenge these as it is a legitimate participant in them. In the third, Alice and Bob accept with member lists containing only each other, while the adversary knows the keys it replayed into them. Challenging either breaks key indistinguishability. The third exchange also leaves two sessions that accept with all-honest member lists yet hold differing keys, breaking match security. Attack 2 is similar: Bob's and Carol's sessions accept with all-honest member lists while holding a key the adversary obtained from an earlier meeting.

In Attack 3, the adversary records the public keys of an honest session for Alice and for Bob, then brute-forces a pair of public keys of its own that makes the two compute the same code. Each accepts with a member list containing only the other honest party, so either may be challenged, yet the exchange is really with the adversary who holds the real key, breaking key indistinguishability.

In Attack 4, Alice's and Bob's sessions each accept believing the other has a matching code, when in fact their codes differ. Neither has a matching session for the honest party in

[13] This means that keys from different protocol executions, as determined by the key context, must be independent.

its member list, so authentication breaks in both directions. As both member lists contain only honest parties, both sessions remain challengeable even though the adversary is part of the key exchange, thus key indistinguishability is broken. Attack 5 breaks the same two properties one-sidedly.

4 Constructing a Secure Human-to-Human Group Key Agreement

Our aim is to present a simple construction demonstrating that the security code method *can* be used to build a secure protocol, providing we avoid the many subtle security pitfalls of the in-band setting (like those discussed in Section 1.4). Thus, we proceed to construct a group key agreement protocol which implements the *security code method*, and prove that it securely realizes human-to-human group key agreement.

As we saw in Section 1.2, real-world implementations tend to follow a two-phase architecture: (1) an initial unauthenticated key exchange between clients, followed by (2) a human-driven verification ceremony, in which parties verify that they agree on (some function of) the key exchange transcript. We capture this familiar approach through a generic composition of an underlying key exchange protocol and security code verification ceremony.[14] Figure 3 is a sequence diagram of the full protocol, including the component sub-protocols.

In the first stage, the client executes an instance of the group key agreement protocol, from which it expects a secret key k, a group size n, and a key context *kid*. Intuitively, the key agreement should guarantee that the resulting key is secret, under the proviso that $(n-1)$ other honest matching sessions exist, having computed the same key context and completed without failure. The key context identifies a particular execution of the protocol and, thus, allows the clients to determine whether they are a part of the same key exchange by comparing their contexts *after* the key exchange has completed; hence, we say that these protocols are "post-authenticated." Appendix C formalizes such protocols under the moniker of Post-Authenticated Group Key Agreement (POST-GKA).

In the second stage, the client initiates a security code verification ceremony, whose role it is to post-authenticate the key agreement, i.e., to identify the $(n-1)$ other people whose clients were part of the key agreement session. To do so, the client provides the user with the key context *kid*, i.e., the security code, and asks them to determine a list of people who have the same security code. When the user has collected such a member list, *mem*, with the same number of users as the expected group size output by the key exchange, n, the ceremony is considered to have completed successfully. Intuitively, the ceremony should guarantee that any honestly acting person who appears in the member list has been provided the same key context from one of their post-authenticated key agreement sessions. We call this a Group-Value Verification Ceremony, and formalize it in Appendix D.

The security of this composition, therefore, relies on the security code both uniquely identifying the key exchange instance and capturing the cryptographic identities of its members. This ensures that, once the verification ceremony has completed with a member list of the same size as the key exchange, our composition can determine that there is a one-to-one mapping from the cryptographic identities of the key exchange participants to the membership output by the verification ceremony. As we will soon see, it suffices to match the number of keys to the number of participants without explicitly determining a matching between them, i.e., clients need not determine which cryptographic key represents which member.

The composition, which realizes the *Security Code Method*, is denoted SCM. It is precisely specified by the state machine of Figure 6 in Appendix F.1, where we also determine its security as a function of the underlying protocol and ceremony.

4.1 Commit-and-Reveal Leader-Based Group Key Agreement

We now describe our post-authenticated group key agreement protocol, which we term Commit-and-Reveal Leader-based Group Key Agreement (crLGKA). The top half of Figure 3 (demarcated with a turquoise dashed line and labeled Π_{crLGKA}) provides a sequence diagram; the state machine is in Appendix F.2, Figure 7.

To start, each participant generates two ephemeral key pairs, one for a digital signature scheme and the other for a public-key encryption scheme; fresh pairs are generated whenever a new session is initialized. Each session additionally generates a random nonce, r, and a commitment to it, $\mathsf{H}_{com}(r)$. The public parts of each key pair are sent to the service provider, accompanied by the commitment. Sessions expect to receive information about the group they will form from either the application layer above, or a third-party (as is the case when a service provider schedules the meeting on the participants' behalf). We denote this role as "service provider" in Figure 3, with the understanding that these may be adversarially controlled (as is reflected in our formalisms). Thus, each session waits for a message describing the group it will join and its role within that group.

A group creation message $(\texttt{create}, \mathit{pks}, \mathit{coms})$ designates a session as the leader of a new group with participant keys *pks* and commitments *coms*. The first entry $\mathit{pks}[0]$ must be the signature verification key generated by the leader session and, similarly, $\mathit{coms}[0]$ must be the commitment published by the leader session. The remaining entries are the public encryption keys of the remaining members, alongside their respective commitments (such that the i-th key was generated by the same session that generated the i-th commitment), and

[14] This approach was proposed for the *out-of-band* setting, alongside a similar generic composition result in [46] for two-party key exchange, and extended to the group key exchange in [35].

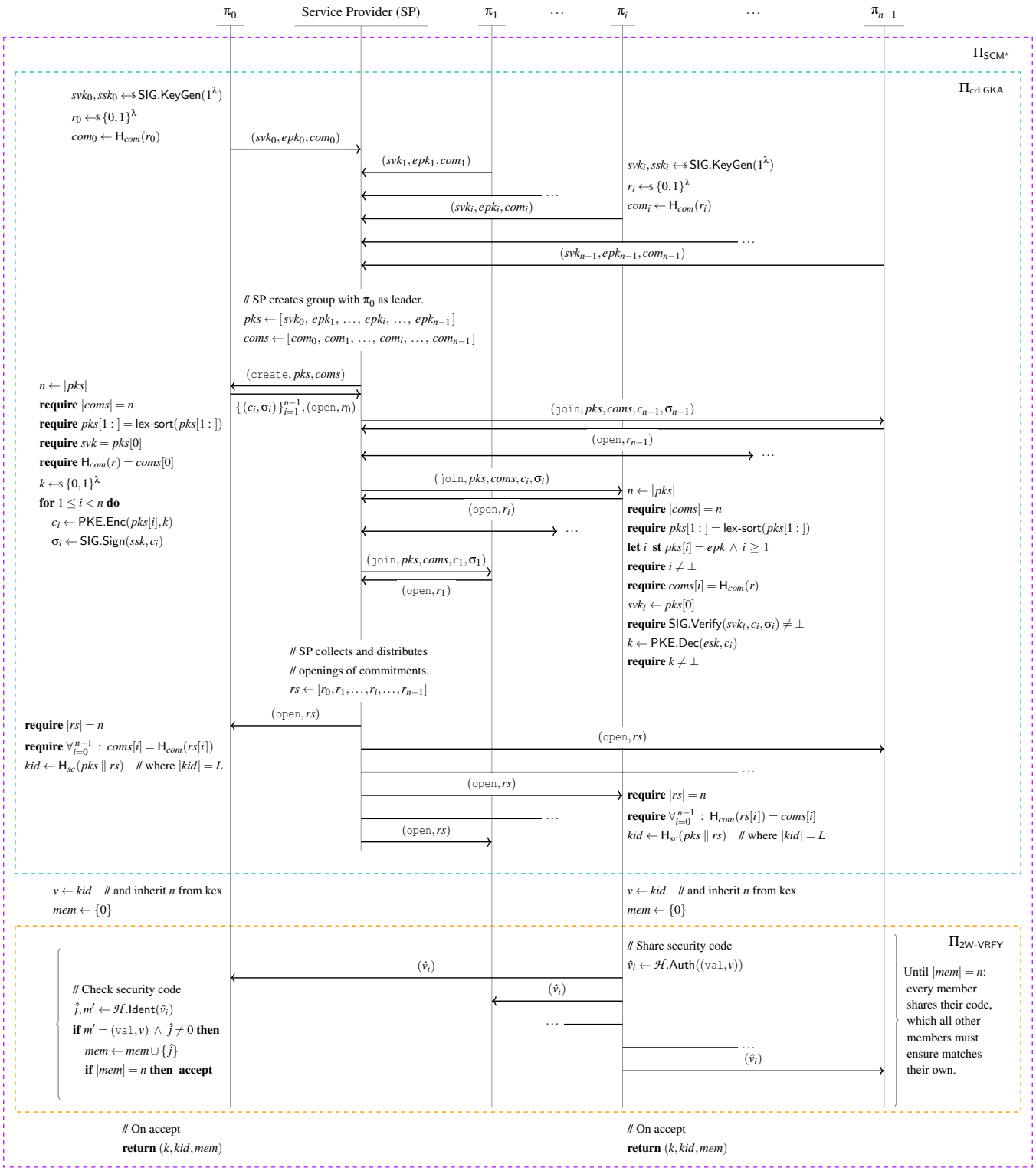

Figure 3: Our human-to-human group key agreement protocol, in which n sessions compute a shared symmetric key, k, and agree upon the group membership, *mem*. A turquoise dashed line encases the post-authenticated group key agreement, crLGKA; an orange dashed line encases the verification ceremony, 2W-VRFY; and a purple dashed line encases their composition, SCM*.

must be in lexicographic order with respect to the public encryption keys. After checking that this is the case, the leader takes note of *pks* and *coms* before sampling a fresh symmetric key for the group. It encrypts this key for each public encryption key in $pks[1:]$ (i.e., for each follower; recall that $pks[0]$ is the leader's own signature verification key, not an encryption key). The resulting ciphertexts are individually signed with the leader's secret signing key and returned to the service provider for distribution alongside the leader's nonce r that was used to generate its commitment.

A join message of the form $(\texttt{join}, pks, coms, c, \sigma)$ instructs a session to join the group as a non-leader; the leader is designated by the verification key $svk_l := pks[0]$. The session ensures that the remaining public encryption keys are in lexicographic order, that its own key is present in *pks*, and that its commitment is in *coms* at the same index as its public encryption key in *pks*. Providing these checks pass, the session verifies the ciphertext-signature pair (c, σ) against the leader's verification key svk_l, before decrypting the ciphertext to determine the shared secret, k. If all steps succeed, the session takes note of the leader verification key, the participant key list, the commitments and the shared secret before returning its nonce r to the service provider.

Before the key agreement is considered a success, all sessions await a message from the service provider, $(\texttt{open}, rs)$, that reveals a valid opening for each commitment in *coms*. Once a session receives and validates these, it sets its key context to $\mathsf{H}_{sc}(pks \| rs)$ and status to `accept`; if the given openings are not valid, it sets its status to `reject`.

4.2 Two-Way Verification Ceremony

A group-value verification ceremony, formalized in Appendix D, aims to capture the process of people working together to discover the group of people they share the same value with. At the start of the ceremony, each person is provided with an input value, v. The participants then communicate amongst themselves in order to *individually* determine the following: "which people do I share the value v with?" If successful, each participant outputs a list of party identifiers, *mem*, indicating the people they believe are in their group. These identifiers are party-local, so they may not be consistent among the different members.

We utilize a *two-way* verification ceremony in our construction: every participant must speak the code directly. This avoids attacks that exploit implicit confirmations, or which replay confirmation messages (providing that security codes are unique; see Appendix D, *Limitations*). Each party is given a value v and an expected group size n upon session initialization; they immediately create a set *mem* of local identifies for parties they have identified thus far, initializing it to $\{0\}$ (indicating themselves with the value 0). Next, they call $\mathcal{H}$.Auth on $(\texttt{val}, v)$, and return the resulting human-authenticated message to the adversary. After this, they start listening for messages (bit strings) $\hat{m}$; for each, they call $\mathcal{H}$.Ident to extract a local speaker identifier $\hat{j}$ and the corresponding message m'. If m' matches the input value v, they add $\hat{j}$ to *mem*. The ceremony starts in a `running` state; once $|mem|$ reaches n, it transitions to `accept`. If any of the above steps fails, it sets its status to `reject` and stops listening.

The bottom half of Figure 3 (labeled $\Pi_{\mathsf{2W\text{-}VRFY}}$) illustrates the two-way verification ceremony. For the corresponding state machine, see Appendix F.3, Figure 8.

4.3 A Secure Composition

We now provide a concrete instantiation of our compositional protocol and ceremony, utilizing the underlying key exchange, crLGKA, and verification ceremony, 2W-VRFY, just developed. We denote this instantiation as SCM*; Figure 3 shows the full composed construction. The composition results of Appendix F.1 along with our analyses of crLGKA and 2W-VRFY (Appendix F.2 and Appendix F.3, respectively), yield the following theorems.

Theorem 1 (Authentication of SCM*)**.** *For any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathsf{SCM}^\star,\lambda,L,n_p,n_s}_{\mathsf{Auth}}(\mathcal{A}) \;=\; 0.$$

This follows directly from Theorems 4 and 9. □

Theorem 1 suggests that our authentication guarantee is perfect, which may seem surprising; we discuss this further in Appendix F.3.1, *Perfect authentication?*.

Theorem 2 (Match Security of SCM*)**.** *For any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{B}$ and $\mathcal{C}$ for which*

$$\mathsf{Adv}^{\mathsf{SCM}^\star,\lambda,L,n_p,n_s}_{\mathsf{Match}}(\mathcal{A}) \;\le\; \frac{(n_s)^2}{2^L} + \frac{(n_{sc})^2}{2^\lambda} + 2\cdot\frac{(n_{com})^2}{2^\lambda} + n_s\left[\mathsf{Adv}^{\mathsf{SIG},\lambda,n_s}_{\mathsf{EUF\text{-}CMA}}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{PKE},\lambda,n_s}_{\mathsf{Correct}}(\mathcal{C})\right].$$

This follows directly from Theorems 5, 7, 9 and 10. □

Theorem 3 (Key Indistinguishability of SCM*)**.** *For any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ for which*

$$\mathsf{Adv}^{\mathsf{SCM}^\star,\lambda,L,n_p,n_s}_{\mathsf{KeyInd}}(\mathcal{A}) \;\le\; \frac{(n_s)^2}{2^L} + \frac{(n_{sc})^2}{2^\lambda} + 2\cdot\frac{(n_{com})^2}{2^\lambda} + n_s\left[\mathsf{Adv}^{\mathsf{SIG},\lambda,n_s}_{\mathsf{EUF\text{-}CMA}}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{PKE},\lambda,n_s}_{\mathsf{Correct}}(\mathcal{C})\right] + n_s\cdot\mathsf{Adv}^{\mathsf{PKE},\lambda,n_s}_{\mathsf{IND\text{-}CCA}}(\mathcal{D}).$$

This follows directly from Theorems 6 to 9. □

In the bounds above, L is the length of the security code, λ is the security parameter, n_p is the number of parties, and

n_s is the total number of sessions. We require H_{com} to be modeled as a random oracle O-RO_{com} (handling at most n_{com} queries) and H_{sc} as a random oracle O-RO_{sc} (handling at most n_{sc} queries). Further, $\mathsf{Adv}^{\mathsf{SIG},\lambda,n_\sigma}_{\mathsf{EUF\text{-}CMA}}$ refers to the existential-unforgeability under adaptive chosen message attack (EUF-CMA security) [30, Def 13.2] of the SIG signature scheme, where n_σ the maximum number of signatures per signing key. $\mathsf{Adv}^{\mathsf{PKE},\lambda,n_m}_{\mathsf{Correct}}$ and $\mathsf{Adv}^{\mathsf{PKE},\lambda,n_m}_{\mathsf{IND\text{-}CCA}}$ refer, respectively, to the correctness (i.e., probability of incorrect decryption)[15] and indistinguishability of encryptions under chosen-ciphertext attack (IND-CCA) [30, Def 12.8] of the PKE public-key encryption scheme, where n_m is the maximum number of queries cumulatively made to the encryption and decryption oracles. Finally, each bound requires that $\mathcal{H}$ implements the idealized human authentication functionality of Section 2.

Colliding codes. The first term in the match security and key indistinguishability bounds, $(n_s)^2/2^L$, captures cases such as the above, where two groups of honest parties compute distinct secret keys, but arrive at the same security code. It also captures more powerful attacks, whereby an attacker exploits such collisions to insert adversarial keys into the group. We highlight that this term is solely determined by the number of sessions (globally) and the length of the security code. It is affected by neither the computational effort available to the adversary, nor the number of random oracle queries they can make, so in that sense it is a *statistical security term*. Intuitively, thanks to the commit-and-reveal scheme we employ, this protocol is not amenable to offline bruteforce attacks: every attempt by an adversary to cause a collision requires them to instantiate a new honest session. This is, of course, similar to the approach taken (and results) of similar work (e.g., [57, Section 5]). It is, nonetheless, an essential component of protocols that use short security codes.

How short is *too* short? To understand how to choose the security code length for our construction, consider the term $(n_s)^2/2^L$ that directly translates to the security code length.[16] Our construction achieves 40 bits of statistical security if the security code length $L = 60$ bits (≈ 19 digits) and the platform allows ≈ 1 thousand sessions during the lifetime of the system; extending it to $L = 80$ bits (≈ 25 digits) lets our construction handle up to ≈ 1 million sessions during its lifetime. For stronger security of 80 bits with up to ≈ 1 million sessions, we would need $L = 120$ bits (≈ 37 digits), and to scale up to ≈ 1 billion sessions, $L = 140$ bits (≈ 43 digits). Alphanumeric codes can be used to reduce the length, but may be more amenable to human error.

How short is short *enough*? At a conservative one second per digit, a 25-digit code would take 25 seconds to read. Since our two-way ceremony asks every participant to read the code, the ceremony grows linearly in the size of the meeting, with a ten person meeting requiring 4 minutes and 10 seconds for the ceremony. It is important to consider whether asking users to dedicate this much time to the ceremony is realistic or viable. Codes of this length are nonetheless deployed in practice, despite their unarguably poor usability.

[15] We define this correctness game implicitly from the probabilistic correctness definition of PKE (as in [30, Def 12.1]).

[16] We assume a standard bit length is chosen for the computational security parameter, which is required to get security from the underlying cryptographic building blocks (signature scheme, PKE and commitment scheme).

5 Conclusions

We have identified Human-to-Human Group Key Agreement (H2H-GKA), a type of protocol that captures E2EE meetings between human users, rather than their devices. Such protocols aim to provide security guarantees without assuming existence of any global identification scheme or that meetings be pre-scheduled. Remarkably, even if only one participant in a group recognizes all the others, we still provide security for that participant. Our H2H-GKA is defined in a idealized *human authentication* model. Our second contribution is defining an idealized functionality that captures people's ability to authenticate messages and, upon receiving such a message, identify consistently who produced it and what the message content is. Finally, we present a simple and practical construction, which we prove to be a secure H2H-GKA in this idealized model. We also discuss how to set concrete parameters for our construction.

Our human authentication functionality makes a variety of normative assumptions about how people communicate. Rather than interpreting it as a prescriptive description of how people authenticate one another, we encourage the reader to interpret it as a collection of properties that existing deployments of the security code method seem to rely on. For example, these requirements may fail to capture people with vision, hearing, or cognitive impairments, who might interpret messages differently. By clearly specifying the assumptions underlying this approach, the human authentication functionality could help guide research into accessibility and/or deepfake detection tools compatible with the in-band security code approach.

Unfortunately, numerous studies have demonstrated that these techniques present usability problems even in the most ideal conditions [29, 32, 53]. In-band usage of the security code method seems to require a burdensome verification ceremony, moreso than in the out-of-band case. We emphasize that we do not claim the security code method to be optimal: where users have well-known identifiers, alternatives such as those utilizing key transparency may well be preferable. On the other hand, this approach enables secure meetings with minimal infrastructure and trust assumptions, while supporting a wide variety of use-cases, such as one-way authenticated meetings, as we discussed in Section 1.2. Indeed, the in-band approach is most compelling precisely when such identifiers, and out-of-band channels, are unavailable. Perhaps an alterna-

tive approach is possible—one that enables users to connect securely with minimal manual effort, while simultaneously avoiding the need for a trusted third-party or out-of-band cooperation. Our definition can provide a benchmark with which to evaluate, and understand, the security tradeoffs involved in moving to alternative approaches, such as those utilizing key transparency.

6 Acknowledgments

We thank Douglas Stebila for his contributions throughout this work. In particular, for our numerous discussions on the design of both the formalisms and human-authentication ideal functionality. We also thank Julia Len for valuable early stage discussions, and Ryan Turner for his support and for insightful discussions regarding real-world deployability. Daniel Jones completed part of this research during an internship at Microsoft Research, and was additionally supported by UKRI Grant EP/S021817/1 as part of the Centre for Doctoral Training in Cyber Security for the Everyday at Royal Holloway. This paper was edited for grammar using Claude Code.

References

[1] Michel Abdalla. Security Analysis of Olvid's SAS-based Trust Establishment Protocol. Cryptology ePrint Archive, Paper 2020/808, 2020. URL: https://eprint.iacr.org/2020/808.

[2] Mashari Alatawi and Nitesh Saxena. VOICE-ZEUS: Impersonating Zoom's E2EE-Protected Static Media and Textual Communications via Simple Voice Manipulations. *arXiv preprint arXiv:2310.13894*, 2023.

[3] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In *Annual International Cryptology Conference*, pages 248–277. Springer, 2020.

[4] Apple. Advancing iMessage security: iMessage Contact Key Verification. https://security.apple.com/blog/imessage-contact-key-verification/, 2023. Accessed: 2026-02-03.

[5] Karen Azari and Andreas Ellison. Tighter provable security for TreeKEM. In Marc Fischlin and Veelasha Moonsamy, editors, *ACNS 2025: 23rd International Conference on Applied Cryptography and Network Security, Part I*, volume 15825 of *Lecture Notes in Computer Science*, pages 572–595, Munich, Germany, June 23–26, 2025. Springer, Cham, Switzerland. doi:10.1007/978-3-031-95761-1_20.

[6] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. RFC 9420, July 2023. URL: https://www.rfc-editor.org/info/rfc9420, doi:10.17487/RFC9420.

[7] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155, Bruges, Belgium, May 14–18, 2000. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-45539-6_11.

[8] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press. doi:10.1145/168588.168596.

[9] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1994. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-48329-2_21.

[10] Josh Blum, Simon Booth, Brian Chen, Oded Gal, Maxwell Krohn, Julia Len, Karan Lyons, Antonio Marcedone, Mike Maxim, Merry Ember Mou, Armin Namavari, Jack O'Connor, Surya Rien, Miles Steele, Matthew Green, Lea Kissner, and Alex Stamos. E2E Encryption for Zoom Meetings. https://github.com/zoom/zoom-e2e-whitepaper/blob/v4.7/zoom_e2e.pdf, 2025. Accessed: 2026-01-13.

[11] Jens-Matthias Bohli, María Isabel González Vasco, and Rainer Steinwandt. Secure group key establishment revisited. *International Journal of Information Security*, 6(4):243–254, 2007.

[12] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Provably authenticated group Diffie-Hellman key exchange – the dynamic case. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 290–309, Gold Coast, Australia, December 9–13, 2001. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-45682-1_18.

[13] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Group Diffie-Hellman key exchange se-

cure against dictionary attacks. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 497–514, Queenstown, New Zealand, December 1–5, 2002. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-36178-2_31.

[14] Chris Brzuska, Eric Cornelissen, and Konrad Kohbrok. Cryptographic security of the MLS RFC, Draft 11. Cryptology ePrint Archive, Paper 2021/137, 2021. URL: https://eprint.iacr.org/2021/137.

[15] Jon Callas, Alan Johnston, and Philip Zimmermann. ZRTP: Media Path Key Agreement for Unicast Secure RTP. RFC 6189, April 2011. URL: https://www.rfc-editor.org/info/rfc6189, doi:10.17487/RFC6189.

[16] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000. doi:10.1007/s001459910006.

[17] Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 219. IEEE Computer Society, 2004. URL: https://doi.ieeecomputersociety.org/10.1109/CSFW.2004.24, doi:10.1109/CSFW.2004.24.

[18] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474, Innsbruck, Austria, May 6–10, 2001. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-44987-6_28.

[19] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. SEEMless: Secure end-to-end encrypted messaging with less user-interaction. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1639–1656, 2019.

[20] Cisco. Zero-Trust Security for Webex. https://www.webex.com/content/dam/wbx/us/documents/pdf/Zero_Trust_Security_White_Paper.pdf, 2021. Accessed: 2026-01-14.

[21] Discord. End-to-End Encryption for Audio and Video. https://support.discord.com/hc/en-us/articles/25968222946071, 2025. Accessed: 2026-01-12.

[22] Yevgeniy Dodis, Daniel Jost, Balachandar Kesavan, and Antonio Marcedone. End-to-End Encrypted Zoom Meetings: Proving Security and Strengthening Liveness. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 157–189. Springer, 2023.

[23] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[24] Benjamin Dowling and Britta Hale. Secure messaging authentication against active man-in-the-middle attacks. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 54–70. IEEE, 2021.

[25] Electronic Frontier Foundation. Key verification. https://ssd.eff.org/module/key-verification, 2025. Accessed: 2026-02-04.

[26] Carl Ellison. Ceremony design and analysis. Cryptology ePrint Archive, Paper 2007/399, 2007. URL: https://eprint.iacr.org/2007/399.

[27] Thore Göbel and Daniel Huigens. Proton Key Transparency Whitepaper. https://proton.me/files/proton_keytransparency_whitepaper.pdf, 2024. Accessed: 2026-02-03.

[28] Yuncong Hu, Kian Hooshmand, Harika Kalidhindi, Seung Jin Yang, and Raluca Ada Popa. Merkle^2: A low-latency transparency log system. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 285–303. IEEE, 2021.

[29] Ronald Kainda, Ivan Flechais, and Andrew William Roscoe. Usability and security of out-of-band channels in secure device pairing protocols. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, pages 1–12, 2009.

[30] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC, third edition, 2020. https://doi.org/10.1201/9781351133036.

[31] Mallory Knodel, Sofía Celi, Olaf Kolkman, and Gurshabad Grover. Definition of end-to-end encryption. Cryptology ePrint Archive, Paper 2024/2085, 2024. URL: https://eprint.iacr.org/2024/2085.

[32] Alfred Kobsa, Rahim Sonawalla, Gene Tsudik, Ersin Uzun, and Yang Wang. Serial hook-ups: a comparative usability study of secure device pairing methods. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, SOUPS '09, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1572532.1572546.

[33] Sven Laur, N. Asokan, and Kaisa Nyberg. Efficient mutual data authentication using manually authenticated strings. Cryptology ePrint Archive, Paper 2005/424, 2005. URL: https://eprint.iacr.org/2005/424.

[34] Sven Laur and Kaisa Nyberg. Efficient mutual data authentication using manually authenticated strings. In David Pointcheval, Yi Mu, and Kefei Chen, editors, *CANS 06: 5th International Conference on Cryptology and Network Security*, volume 4301 of *Lecture Notes in Computer Science*, pages 90–107, Suzhou, China, December 8–10, 2006. Springer Berlin Heidelberg, Germany. doi:10.1007/11935070_6.

[35] Sven Laur and Sylvain Pasini. SAS-based group authentication and key agreement protocols. In Ronald Cramer, editor, *PKC 2008: 11th International Workshop on Theory and Practice in Public Key Cryptography*, volume 4939 of *Lecture Notes in Computer Science*, pages 197–213, Barcelona, Spain, March 9–12, 2008. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-78440-1_12.

[36] Sean Lawlor and Kevin Lewi. Deploying key transparency at WhatsApp. https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/, 2023. Accessed: 2026-02-03.

[37] Julia Len, Melissa Chase, Esha Ghosh, Daniel Jost, Balachandar Kesavan, and Antonio Marcedone. Elektra: Efficient lightweight multi-device key transparency. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2915–2929, 2023.

[38] Julia Len, Melissa Chase, Esha Ghosh, Kim Laine, and Radames Cruz Moreno. OPTIKS: An optimized key transparency system. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 4355–4372, 2024.

[39] Harjasleen Malvai, Lefteris Kokoris-Kogias, Alberto Sonnino, Esha Ghosh, Ercan Oztürk, Kevin Lewi, and Sean F. Lawlor. Parakeet: Practical key transparency for end-to-end encrypted messaging. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023.

[40] Mark Manulis. Survey on security requirements and models for group key exchange. Cryptology ePrint Archive, Paper 2006/388, 2006. URL: https://eprint.iacr.org/2006/388.

[41] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. CONIKS: Bringing key transparency to end users. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 383–398, 2015.

[42] Microsoft. Use end-to-end encryption for Microsoft Teams calls. https://support.microsoft.com/en-us/office/use-end-to-end-encryption-for-microsoft-teams-calls-1274b4d2-b5c5-4b24-a376-606fa6728a90, 2025. Accessed: 2026-01-13.

[43] Moni Naor, Lior Rotem, and Gil Segev. Out-of-band authenticated group key exchange: From strong authentication to immediate key delivery. In *1st Conference on Information-Theoretic Cryptography (ITC 2020)*, pages 9–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

[44] Moni Naor, Gil Segev, and Adam Smith. Tight bounds for unconditional authentication protocols in the manual channel and shared key models. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 214–231, Santa Barbara, CA, USA, August 20–24, 2006. Springer Berlin Heidelberg, Germany. doi:10.1007/11818175_13.

[45] Saurabh Panjwani and Achintya Prakash. Crowdsourcing attacks on biometric systems. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 257–269, Menlo Park, CA, July 2014. USENIX Association. URL: https://www.usenix.org/conference/soups2014/proceedings/presentation/panjwani.

[46] Sylvain Pasini and Serge Vaudenay. SAS-based authenticated key agreement. In *Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings 9*, pages 395–409. Springer, 2006.

[47] Martin Petraschek, Thomas Hoeher, Oliver Jung, Helmut Hlavacs, and Wilfried Gansterer. Security and Usability Aspects of Man-in-the-Middle Attacks on ZRTP. *JUCS - Journal of Universal Computer Science*, 14(5):673–692, 2008. URL: https://doi.org/10.3217/jucs-014-05-0673, arXiv:https://doi.org/10.3217/jucs-014-05-0673.

[48] Bertram Poettering, Paul Rösler, Jörg Schwenk, and Douglas Stebila. SoK: Game-based security models for group key exchange. In Kenneth G. Paterson, editor, *Topics in Cryptology – CT-RSA 2021*, volume 12704 of *Lecture Notes in Computer Science*, pages 148–176, Virtual Event, May 17–20, 2021. Springer, Cham, Switzerland. doi:10.1007/978-3-030-75539-3_7.

[49] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema. In *IEEE European Symposium on Security and Privacy, EuroS&P 2018*,

pages 415–429, 2018. doi:10.1109/EuroSP.2018.00036.

[50] Lior Rotem and Gil Segev. Out-of-band authentication in group messaging: Computational, statistical, optimal. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 63–89, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland. doi:10.1007/978-3-319-96884-1_3.

[51] Maliheh Shirvanian and Nitesh Saxena. Wiretapping via mimicry: Short voice imitation man-in-the-middle attacks on crypto phones. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 868–879, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press. doi:10.1145/2660267.2660274.

[52] Signal. What is a safety number and why do I see that it changed? https://support.signal.org/hc/en-us/articles/360007060632, n.d. Accessed: 2026-01-14.

[53] Joshua Tan, Lujo Bauer, Joseph Bonneau, Lorrie Faith Cranor, Jeremy Thomas, and Blase Ur. Can unicorns help users compare crypto key fingerprints? In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, page 3787–3798, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3025453.3025733.

[54] Michael Troncoso and Britta Hale. The bluetooth CYBORG: analysis of the full human-machine passkey entry AKE protocol. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021. URL: https://www.ndss-symposium.org/ndss-paper/the-bluetooth-cyborg-analysis-of-the-full-human-machine-passkey-entry-ake-protocol/.

[55] Nirvan Tyagi, Ben Fisch, Andrew Zitek, Joseph Bonneau, and Stefano Tessaro. VeRSA: Verifiable registries with efficient client audits from RSA authenticated dictionaries. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2793–2807, 2022.

[56] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. SoK: Secure Messaging. In *2015 IEEE Symposium on Security and Privacy*, pages 232–249. IEEE, 2015.

[57] Serge Vaudenay. Secure communications over insecure channels based on short authenticated strings. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 309–326, Santa Barbara, CA, USA, August 14–18, 2005. Springer Berlin Heidelberg, Germany. doi:10.1007/11535218_19.

[58] Théophile Wallez, Jonathan Protzenko, and Karthikeyan Bhargavan. TreeKEM: A modular machine-checked symbolic security analysis of group key agreement in messaging layer security. Cryptology ePrint Archive, Paper 2025/410, 2025. URL: https://eprint.iacr.org/2025/410.

[59] Kevin Warren, Tyler Tucker, Anna Crowder, Daniel Olszewski, Allison Lu, Caroline Fedele, Magdalena Pasternak, Seth Layton, Kevin Butler, Carrie Gates, and Patrick Traynor. "Better Be Computer or I'm Dumb": A Large-Scale Evaluation of Humans as Audio Deepfake Detectors. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 2696–2710, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3658644.3670325.

[60] WhatsApp. About end-to-end encryption. https://faq.whatsapp.com/820124435853543, n.d. Accessed: 2026-02-03.

[61] Zoom. Using end-to-end encryption (E2EE) in Zoom meetings. https://support.zoom.com/hc/en/article?id=zm_kb&sysparm_article=KB0065408, 2025. Accessed: 2026-01-12.

A Preliminaries

A.1 Notation

We use the following notation throughout this work.

Sampling. We use '$x \leftarrow\!\$\ \mathcal{D}$' to denote a value being sampled from the distribution $\mathcal{D}$ and assigned to the variable x. When the right-hand side is a set, rather than a distribution, then $x \leftarrow\!\$\ S$ denotes a value being sampled uniformly from the set S and assigned to the variable x.

Probabilistic algorithms. We use '$x \leftarrow\!\$\ \mathsf{fn}(y)$' to denote variable assignment from the execution of a probabilistic algorithm fn (called with input y) to a variable x.

Lists. We represent lists with the notation '$[a,b,c]$'. List share their semantics with an ordere tuple, and support (a) iteration, i.e., '**for** x **in** $[a,b,c]$ **do**…', (b) indexed access, i.e., '$[a,b,c][1]$' evaluates to b, (c) indexed assignment, i.e., '$[a,b,c][1] \leftarrow d$' results in the list $[a,d,c]$, and (d) slices, i.e., '$[a,b,c][0 \rightarrow 1]$' evaluates to the list $[a,b]$. Slices are inclusive of the endpoint index given.

If a slice reaches out-of-bounds, it returns all the elements it can (without error). The length of a list, say xs, can be computed with $|xs|$. Lists can be concatenated, i.e., '$[a,b]\|[c,d]$' evaluates to $[a,b,c,d]$.

Named tuples. A named tuples can be initialized by providing an entry name alongside each value, i.e., '$tup \leftarrow (x{:=}1, y{:=}2, z{:=}3)$' initializes a named tuple containing three entries named 'x', 'y' and 'z', and taking values 1, 2 and 3 respectively. If a name is not given explicitly, it takes the name of the variable used to initialize it.Individual entries can be accessed (and written to) using the tuple name followed by a dot and the appropriate entry name.

Maps. The '`Map(<init-expr>)`' function creates a data structure mapping labels to values. We represent this internally as a set of label-value tuples. An initialization value '`<init-expr>`' can be provided. Values may be set with '$M[l] \leftarrow v$' and retrieved with '$v \leftarrow M[l]$'. We allow iteration over the values of a map with '**for** $v \in M[\,\cdot\,]$ **do**…', and iteration over the label-value tuples with '**for** $(l,v) \in M$ **do**…'. $|M|$ is the number of label-value tuples in M.

Let expressions. We use let expressions to set the value of a variable based on a set of requirements. The expression '**let** `<var>` **st** `<condition>`' denotes that the variable `<var>` should take a value that satisfies the condition, `<condition>`. If the condition is not satisfiable, or is satisfied by multiple values, the variable is given the value of the error symbol, $\perp$.

A.2 Protocol and Ceremony Definitions

The formalisms we present in this work capture interactive processes that take place between two or more distinct parties. We describe processes between clients (i.e., computers) as *protocols* and processes between people as *ceremonies*. The formalism introduced in Section 3 jointly captures a combined protocol and ceremony, in which case we consider the person and the client they control as a single entity. We define the behavior of the parties using a state machine: an algorithm that takes as input a private state, st, and public message, m, which it processes before returning an updated private state, st', and a public response, m':

$$\Pi(st, m) \rightarrow (st', m').$$

We use the term *session* to refer to a particular execution of the protocol (or ceremony) by a particular *party*. A session is initialized by executing $\Pi(\perp, m)$, which directs a party, say i, to start a new session with the initial message m. The algorithm outputs a state st' and optional response m'.

It proceeds through repeated calls to the state machine algorithm, Π. The first input and output represent its private state st, a named tuple that stores information that is local to this particular party. It must include a status field, $st.x$, within this state. Sessions can rely on the contents of the local state to remain private to them, up-to-date, and uncorrupted. Messages, on the other hand, are generally assumed to be public and untrusted, and may be routed arbitrarily between sessions (capturing an adversarial and unreliable network).

When a session completes execution, it sets its status, $st.x$, to either `accept` or `reject`. The former, `accept`, denotes that the protocol (or ceremony) has completed successfully from the perspective of this session, while the latter, `reject`, indicates that it has not. Sessions in the `accept` state may continue to receive and process messages, but sessions in the `reject` state will not. Note that our syntax for interactive protocols and ceremonies is not able to capture persistent or shared state over multiple executions by a single party.

Our pseudocode descriptions of state machines make use of the following specialized notation.

Assertions. The '**require** `<expr>`' instruction checks the value of `<expr>` and, if it evaluates to **false**, exits the algorithm immediately. All state changes that have occurred during the execution of the algorithm containing the assertion are rolled back. Similarly, any assignments to its return values at the point-of-call are skipped (as if the call never took place). In the case where fresh variables are declared at the call site, they take the value $\perp$.

Pattern matching. Procedures may be defined for particular argument values (or patterns of values), using the syntax '`<arg>:<pattern-or-value>`'. When describing a pattern, the token '$\cdot$' matches any value (including **null**). If a procedure is not defined for all possible values, any cases left undefined execute no instructions and output $\perp$ for all return values. If the interpretation of `<pattern-or-value>` is not clear, it should be described in prose as part of the definition. Pattern matching may also be used when unpacking a tuple, in which case the statement should be interpreted as an assertion that is checked before the unpacking operation is executed. As with assertions, if a pattern matching operation fails, any state changes that have occurred during the current execution of the algorithm are rolled back.

A.3 Security Definitions

A.3.1 Security Experiments

We define a number of distinct security properties for each formalism, with a separate experiment for each property. However, due to the similarity between experiments capturing different properties of the same formalism, we define them using a shared figure. When defining the challenger algorithm, we provide the name of each property (and its respective experiment) within braces, and with color coded highlights. For example, we may write $\{\text{PropertyA}, \text{PropertyB}\}^{\Pi,\Lambda}_{\mathsf{Formalism}}(\mathcal{A})$, in

which case any pseudocode highlighted in yellow applies only to the PropertyA experiment (and pseudocode highlighted in blue applies only to the PropertyB experiment). Any code that is not highlighted applies to all versions of the experiment.

A.3.2 Specialized Notation

We utilize the following specialized notation in our definitions of security experiments.

Oracles. We signal that a particular algorithm represents an oracle by prepending O- to its name. For example, we may write O-Enc as the name of an encryption oracle.

Experiment parameters. We leave the enforcement of usage parameters in an experiment, such as limits on the number of queries, implicit.

Shared state. By default, all algorithms executed by the challenger (i.e., the experiment and its oracles) have access to a shared global state. The adversary only has access to state that is explicitly granted to them through calls to $\mathcal{A}$ or oracles.

Abort. The instruction '**abort**' can be used to end the current experiment early, returning the value 1 to indicate that the adversary has won the experiment.

Assertions within oracles. If an assertion '**require** `<expr>`' within the code defining an oracle fails, then the oracle stops executing immediately, any state changes made during execution of the current oracle are rolled back, and it returns $\perp$. In other words, it silences the query.

B Additional Details on Human-to-Human Group Key Agreement

We provide a prosaic description of the security experiments in Definitions 1 to 3.

Security Experiment. At the start of each experiment, the challenger creates n_p human identities using the human authentication ideal functionality. Before doing so, it gives the adversary an opportunity to (1) choose the encoding and decoding algorithms (see Section 2.2), and (2) select a subset of identities to be adversarially controlled. This allows us to prove security for any valid encoding scheme against an adversary that can command adversarial human identities to act arbitrarily.

Next, the challenger passes control to the adversary, giving it access to the $(\mathcal{H}.\mathsf{Auth}_i, \mathcal{H}.\mathsf{Ident}_i)$ oracles for controlling adversarial human identities, as well as the $\mathcal{H}$.Mapback oracle to allow it to convert between local and global identifiers (Section 2.3). The adversary is given access to oracles O-Init and O-ReqResp to initialize and interact with honest sessions, as well as oracles O-Chall and O-Reveal, which are used to capture the secrecy of keys.

Calling O-Init(i) creates a new session for the i-th party (we allow multiple sessions per party). The challenger initializes a state machine $\Pi_{\mathsf{H2H\text{-}GKA}}$ and internally stores its state st. Finally, it forwards the output message, the session's index within the experiment, and the public part of st (the current status, key context, and membership list) to the adversary.

After a session is created with O-Init, the adversary can interact with it by calling O-ReqResp(i,s,m), which simulates party i's session s (local index) processing a request m. It stores the updated state machine state internally and forwards the output message m', along with the public fields of st, to the adversary.

At the beginning of the experiment, the challenger flips a coin to determine a bit b. For any party i with session s (that has accepted), the adversary can call O-Chall(i,s) to challenge the session, whereby the challenger either returns the real secret key for this session (when $b=0$) or a random replacement (when $b=1$). If a session is challenged, its key context is stored in a set CH. In the key indistinguishability experiment, the adversary wins by correctly guessing the challenge bit. The adversary can only challenge sessions for which $st.mem$ consists of honest parties, as otherwise the adversary may be controlling an honestly participating matching session and would therefore know the key.

The adversary can call O-Reveal(i,s) to reveal the real secret key of a session that has accepted. This marks the key as *weak* by storing its key context in a set WK.

Calls to O-Chall and O-Reveal are restricted and related to each other, as we will detail below when discussing key indistinguishability. The purpose of the sets CH and WK is to enforce these restrictions.

C Post-Authenticated Group Key Agreement

Syntax. A post-authenticated group key agreement is an interactive protocol (see Appendix A.2) in which a group of sessions work together to compute a shared secret key and an associated key context. It is defined by a state machine $\Pi_{\mathsf{POST\text{-}GKA}}(st,m) \to (st',m')$.

Sessions start with no initial input to guide them, and proceed by processing and responding to requests from one another (transmitted over an insecure network). Throughout, the state machine maintains a state, st, containing the following public and private fields:

- [public] $st.x$ storing the current status,
- [public] $st.kid$ storing the key context,
- [public] $st.n$ storing the expected group size, and
- [private] $st.k$ storing the shared key.

A session indicates success by setting its status to `accept`. This indicates that, from the perspective of this party, the

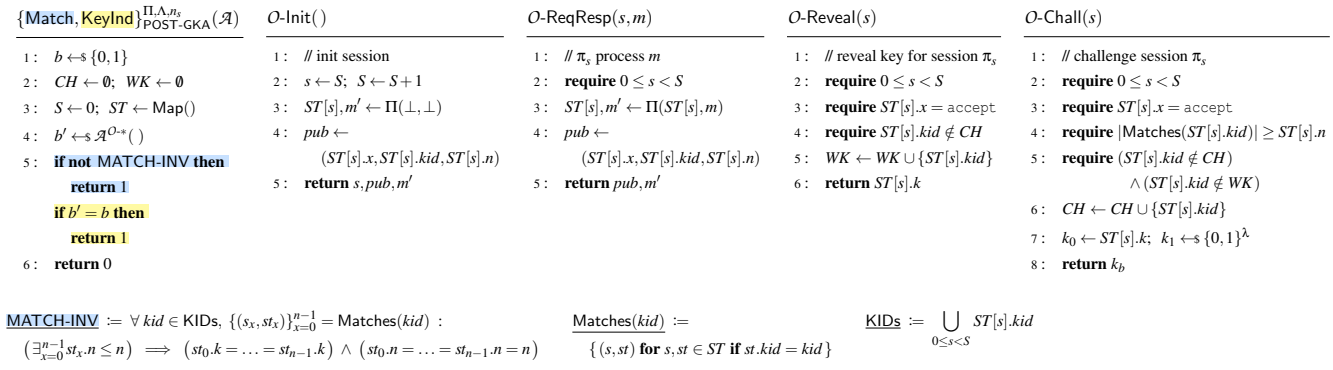

Figure 4: Security experiments defining the match security and key indistinguishability of Post-Authenticated Group Key Agreement protocols.

group agreed on a shared secret key $st.k$ with key context $st.kid$, under the proviso that $(st.n-1)$ other honest sessions accepted with a matching key context equal to $st.kid$. On the other hand, a session indicates failure by setting its status to `reject`.

Security. We capture the security of post-authenticated group key agreement through two properties: match security and key indistinguishability. We define each of these properties in turn, with reference to the security experiments described in Figure 4.

The two experiments follow a similar structure. The adversary is given access to four oracles: O-Init, O-ReqResp, O-Reveal, and O-Chall. $O\text{-Init}(m) \to (s,m')$ allows it to trigger the initialization of a new honest session and pass an initial message m. The challenger will set up the state for this session, run the initialization routine, and forward the resulting message along with the session's index within the experiment back to the adversary. The adversary interacts with each session through the $O\text{-ReqResp}(s,m) \to m'$ oracle, where the challenger simulates session s processing a request m and forwards on the response m' to the adversary. The challenger returns the response message, m', and the session's public state pub at the end of the O-Init and O-ReqResp oracles.

Match security. Intuitively, a secure protocol should ensure that matching sessions, those that are part of the same protocol execution, compute the same key and group size. As described above, the protocol outputs a key context that the application layer may use to determine which sessions are part of the same execution, and provides security if the correct number of honest sessions have computed a matching key context. We are, thus, able to mirror these preconditions in our definition of match security, resulting in a weaker security notion than H2H-GKA match security in Figure 2.

Specifically, we ask that for any key context that is computed by an honest session, if at least one honest matching session has computed a group size no larger than the number of honest matching sessions, then all such sessions have computed the same key and group size. We formalize match security as follows.

Definition 4 (Match Security of POST-GKA)**.** The advantage of an adversary, $\mathcal{A}$, against the match security, as defined by Match in Figure 4, of a post-authenticated group key agreement protocol, Π, is

$$\mathsf{Adv}^{\Pi,\Lambda,n_s}_{\mathsf{Match}}(\mathcal{A}) := \Pr\left[\mathsf{Match}^{\Pi,\Lambda,n_s}_{\text{POST-GKA}}(\mathcal{A}) \mapsto 1\right].$$

Here, Λ are the parameters used to instantiate Π, and n_s is the maximum number of sessions.

Key indistinguishability. We define key indistinguishability similarly to the human-to-human group key agreement formalism above. At the beginning of the experiment, the challenger flips a coin to determine the challenge bit b. The adversary wins the key indistinguishability experiment by correctly guessing this challenge bit, which the adversary attempts to learn by orchestrating execution of the protocol (using O-Init and O-ReqResp), as well as asking for challenge keys (using O-Chall) or revealing real keys (using O-Reveal).

The adversary may ask for challenges through $O\text{-Chall}(s)$ calls, whereby the challenger either returns the real key of session s (when $b=0$) or a random replacement (when $b=1$). Only sessions that have accepted with a group size upper-bounded by the number of honest matching sessions in the experiment may be challenged. This captures the guarantees expected from the application layer which, in turn, ensure that no adversarially controlled session is an *expected participant* in the protocol execution.

The adversary may also reveal the real key of an accepting session by calling $O\text{-Reveal}(s)$. Keys which have been revealed are then marked as *weak*. No sessions sharing a key context with an already challenged key, or marked as weak, may be challenged.

We formalize key indistinguishability as follows.

Definition 5 (Key Indistinguishability of POST-GKA)**.** The advantage of an adversary, $\mathcal{A}$, against the key indistinguishability, as defined by KeyInd in Figure 4, of a post-authenticated group key agreement protocol, Π, is

$$\mathsf{Adv}^{\Pi,\Lambda,n_s}_{\mathsf{KeyInd}}(\mathcal{A}) := \Pr\left[\mathsf{KeyInd}^{\Pi,\Lambda,n_s}_{\mathsf{POST\text{-}GKA}}(\mathcal{A}) \mapsto 1\right] - \frac{1}{2}.$$

Here, Λ are the parameters used to instantiate Π, and n_s is the maximum number of sessions.

D Group-Value Verification Ceremony

Syntax. A group-value verification ceremony is an interactive ceremony (see Appendix A.2) in which people organize into groups based upon a shared token, or value. It is defined by a state machine $\Pi^{\mathcal{H}.\mathsf{Auth}_i,\mathcal{H}.\mathsf{Ident}_i}_{\mathsf{VRFY}}(st,m) \to (st',m')$ that has oracle access to party i's human authentication interfaces.

Sessions are given a value, v, and an expected group size, n, as their initial input. It is this value that determines the group they are a part of. They proceed by processing and responding to requests from one another (transmitted over an insecure network), while each builds a list of the members of their group (i.e., a list of people sharing the same value as them). The formalism utilizes a human authentication functionality, such as the one defined in Section 2, to capture human-authenticated messages and their identities. Throughout the ceremony, the state machine (or, rather, the *person* executing it) maintains a state, st, containing the following public information:

- [public] $st.x$ storing the current status,
- [public] $st.v$ storing the value,
- [public] $st.n$ storing the expected group size,
- [public] $st.mem$ storing the group membership list.[17]

A session indicates success by setting its status to `accept`. This indicates that, from the perspective of this party, the people whose identities are in $st.mem$ are members of the group defined by the shared value $st.v$. On the other hand, a session indicates failure by setting its status to `reject`.

We do not expect the verification ceremony to always correctly identify all members of a group. Instead, our authentication property requires that any person in the member list, who is following the ceremony honestly truly does have a session with the same value. Our match security property asks for stronger guarantees, that all sessions with the same value agree on the members of the group, but only does so in situations where the adversary has allowed the ceremony to complete without sufficient tampering. We proceed to define these two properties formally in the paragraphs that follow.

Security. We capture the security of group-value verification ceremonies through two properties: match security and authentication. We define each of these properties in turn, with reference to the security experiments described in Figure 5.

The two experiments follow a similar structure. The adversary is able to create up to n_p honest parties through the O-Gen oracle. The setup and initialization of the human authentication functionality follows identically to that of the H2H-GKA definition in Figure 2. Intended to capture the adversary's capability of having people participate in verification ceremonies on its behalf, the adversary is able to create adversarial parties by calling directly into the $\mathcal{H}$ functionality. These adversarial parties are able to create and identify human-authenticated messages in the same manner that adversarial parties may. The adversary may initiate up to n_s sessions in total throughout the experiment (via the O-Init oracle). When initiating a session, the adversary provides the global index of the party that the challenger is initiating the session on behalf of, as well as the value it is verifying, and receives the party-local index of the new session accompanied by any initial outputs. The O-ReqResp oracle allows the adversary to interact with and orchestrate each session's execution of the protocol. We expect the adversary to directly run the verification ceremony on behalf of adversarial parties and, thus, we restrict calls to O-Init and O-ReqResp to honest parties only.

Match security. In idealized conditions, we would expect a ceremony to enable a group of people to find one another, and to arrive upon an accurate consensus on who is part of their group. We capture these requirements through match security. Intuitively, we ask that any set of accepting sessions with matching values agree on the group member list (when party-local identifiers are converted to global identifiers) providing that: (a) there does not exist an adversarially-controlled human identity in any of the accepting sessions' member lists, and (b) these sessions agree on a group size that matches the number of matching sessions in the experiment. Note that, unlike Byzantine consensus, this match security definition only requires consensus when all participants act honestly. It is easy to imagine a strengthening of this definition that does allow for a dishonest subset of members (and the additional effort from participants this may entail).

We capture match security formally as follows.

Definition 6 (Match Security of VRFY)**.** The advantage of an adversary, $\mathcal{A}$, against the match security, as defined by Match in Figure 5, of a verification ceremony, Π, is

$$\mathsf{Adv}^{\Pi,\Lambda,n_p,n_s}_{\mathsf{Match}}(\mathcal{A}) := \Pr\left[\mathsf{Match}^{\Pi,\Lambda,n_p,n_s}_{\mathsf{VRFY}}(\mathcal{A}) \mapsto 1\right].$$

Here, Λ are the parameters used to instantiate Π, n_p is the number of parties, and n_s is the maximum number of sessions.

Authentication. We expect the ceremony to guarantee some properties under all conditions (regardless of whether adversarially-controlled users are part of the group, or whether

[17] The member list $st.mem$ must be initialized with a single entry, 0, to indicate that the user themselves is part of the group.

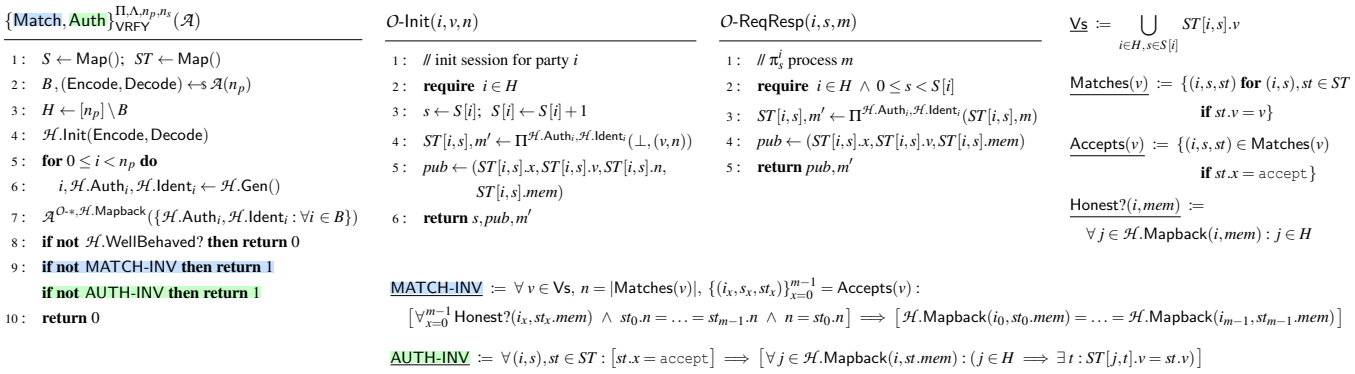

Figure 5: Security experiments defining the match security and authentication of Group-Value Verification ceremonies (with respect to a human authentication ideal functionality, $\mathcal{H}$).

there is an accurate consensus on the group size). We say that a verification ceremony provides authentication if it guarantees that whenever an honest session believes a particular party to have a session with a matching value, and that party is honest, then such a matching session exists. We capture authentication formally as follows.

Definition 7 (Authentication of VRFY)**.** The advantage of an adversary, $\mathcal{A}$, against the authentication, as defined by Auth in Figure 5, of a verification ceremony, Π, is

$$\mathsf{Adv}^{\Pi,\Lambda,n_p,n_s}_{\mathsf{Auth}}(\mathcal{A}) \coloneqq \Pr\left[\mathsf{Auth}^{\Pi,\Lambda,n_p,n_s}_{\mathsf{VRFY}}(\mathcal{A}) \mapsto 1\right].$$

Here, Λ are the parameters used to instantiate Π, n_p is the number of parties, and n_s is the maximum number of sessions.

Limitations. This formalism is not able to capture situations where two semantically distinct groups share the same group value. This places limitations on the composed protocol, requiring the security codes that the key exchange protocol outputs to be globally unique; in turn, requiring that the hash function used to compute the security code never produces a collision. Alternative formulations may be able to overcome this limitation by preconditioning group membership on implicitly agreed upon context. For example, if we assume that participants agree on the current date, we need only ask for security codes to be unique within each day.

E Properties of Human Authentication

In this section, we formalize and prove four properties of our human authentication ideal functionality that we alluded to in Section 2.4. These properties hold as long as the *WellBehaved*? flag is true, i.e., both *InconsistentDecoding*? and *CheatWithPrediction*? are false. First, however, we need the following helpful observation regarding $\mathcal{H}.\mathsf{Mapback}$.

Correctness of Mapback. $\mathcal{H}.\mathsf{Mapback}$ correctly reflects local identifiers so that $\mathcal{H}.\mathsf{Mapback}(i,\hat{j}) = j$ if and only if $ID[i,j] = \hat{j}$.

Proof. To see this, note first that for any i there cannot be $j \neq j'$, such that $ID[i,j] = ID[i,j'] \neq \emptyset$. This is by construction of $\mathcal{H}.\mathsf{Ident}_i$, which always sets each new $ID[i,k]$ to the number of previous non-empty $ID[i,\cdot]$ values; this means that this index is always increasing and cannot repeat. This implies that if $ID[i,j] = \hat{j}$, then $\mathcal{H}.\mathsf{Mapback}(i,\hat{j}) = j$. The other direction follows by construction of $\mathcal{H}.\mathsf{Mapback}$. □

The four properties of $\mathcal{H}$ can be stated and proven as follows.

Correct message extraction. As long as *WellBehaved*? is true, authenticated messages produced by $\mathcal{H}.\mathsf{Auth}_i(m)$ are always correctly decoded by $\mathcal{H}.\mathsf{Ident}_j$ to m.

Proof. As $\mathcal{H}.\mathsf{Auth}_i(m)$ produced $\hat{m}$, it ensured $\mathsf{Decode}(\hat{m})$ returns (i,m) and $(i,m,\hat{m}) \in AM$; otherwise *InconsistentDecoding*? would be set. Since *CheatWithPrediction*? is not set, $\hat{m}$ is not added to RM. This guarantees that $\mathcal{H}.\mathsf{Ident}_j$ will always return m. □

Correct identification. As long as *WellBehaved*? is true, any party j will identify i correctly from $\hat{m}$ according to j's local indexing scheme if $\hat{m} = \mathcal{H}.\mathsf{Auth}_i(m)$. Specifically, if $\mathcal{H}.\mathsf{Ident}_j(\hat{m}) = (\hat{i},m')$, then $\mathcal{H}.\mathsf{Mapback}(j,\hat{i}) = i$.

Proof. $\mathcal{H}.\mathsf{Auth}_i(m)$ always checks that $\mathsf{Decode}(\hat{m})$ will return the right i. If $\mathcal{H}.\mathsf{Ident}_j(\hat{m})$ does not output $\bot,\bot$, then it will output the local identifier $\hat{i}$ corresponding to i output by Decode, i.e., $ID[j,i] = \hat{i}$. By *Correctness of Mapback*, $\mathcal{H}.\mathsf{Mapback}(j,\hat{i}) = i$. □

Unforgeability. As long as *WellBehaved*? is true, for any j and $\hat{m}$, if $\mathcal{H}.\mathsf{Ident}_j(\hat{m}) = (\hat{i},m)$ and $i = \mathcal{H}.\mathsf{Mapback}(j,\hat{i})$, then there was a previous call to $\mathcal{H}.\mathsf{Auth}_i(m)$.[18]

[18] Note that this does not mean that $\mathcal{H}.\mathsf{Auth}_i(m)$ returned the exact authenticated message that is passed to $\mathcal{H}.\mathsf{Ident}_j$. A human-authenticated message may change during transit, through adversary's actions, or through other errors, yet its decoding may remain correct.

Proof. To see this, suppose $\mathcal{H}.\mathsf{Ident}_j(\hat{m})$ outputs $(\hat{i},m)$ and $i = \mathcal{H}.\mathsf{Mapback}(j,\hat{i})$. By construction, $\mathcal{H}.\mathsf{Ident}_j$ will only output $(\hat{i},m)$ if there exists $\hat{m}'$ such that $(z,m,\hat{m}') \in AM$ and if (after $\mathcal{H}.\mathsf{Ident}_j$ executes) $ID[j,z] = \hat{i}$. By *Correctness of Mapback*, this means that $z = \mathcal{H}.\mathsf{Mapback}(j,\hat{i}) = i$, so in fact we have $(i,m,\hat{m}') \in AM$. Finally, note that such a tuple will only be added to AM if there is a call to $\mathcal{H}.\mathsf{Auth}_i(m)$. □

Message and identification consistency. As long as *WellBehaved*? is true, for any $\hat{m}$, either (a) $\mathcal{H}.\mathsf{Ident}_i(\hat{m})$ rejects for every i, or (b) for every i and $(\hat{j},m) = \mathcal{H}.\mathsf{Ident}_i(\hat{m})$, $(\mathcal{H}.\mathsf{Mapback}(i,\hat{j}),m)$ is the same. Repeated calls to $\mathcal{H}.\mathsf{Ident}_i(\hat{m})$ always return the same result.

Proof. First, note that neither condition that causes $\mathcal{H}.\mathsf{Ident}_i(\hat{m})$ to reject depends on the party i, and that the first rejection condition is deterministic (as Decode is deterministic). The second rejection condition cannot change from false to true, because AM cannot lose elements; conversely, it cannot change from true to false without triggering the *CheatWithPrediction*? error flag and setting *WellBehaved*? to false. Therefore, for each party i, $\mathcal{H}.\mathsf{Ident}_i(\hat{m})$ consistently either rejects or accepts.

Now suppose instead $\mathcal{H}.\mathsf{Ident}_i(\hat{m}) = (\hat{j},m)$ and $\mathcal{H}.\mathsf{Ident}_k(\hat{m}) = (\hat{j}',m')$, where possibly $i = k$ (repeated call). It follows from *Unforgeability* that $\mathcal{H}.\mathsf{Auth}_j(m)$ and $\mathcal{H}.\mathsf{Auth}_{j'}(m')$ must have been called successfully and both returned $\hat{m}$. By *Correct identification*, this is possible only if $j = j'$. By *Correct message extraction*, $\hat{m}$ must decode correctly, proving that $m = m'$. □

F Details and Security Analysis of Our Construction

In this section, we provide a detailed description and security analysis of our construction, as well as the key exchange and verification ceremony it is built upon. We, additionally, state and prove the security of each.

F.1 Group Key Agreement with In-band Verification

Figure 6 provides a pseudocode description of our H2H-GKA protocol that we call SCM. It accompanies the prosaic description and sequence diagram in Section 4.

F.1.1 Security Analysis

We now determine the security of SCM as a composition of the underlying post-authenticated group key agreement protocol and group-value verification ceremony.

$\Pi_{\mathsf{SCM}}(st:\perp, m:\perp)$

```
1: x ← kex
2: kst, m' ← Π_POST-GKA(⊥, ⊥)
3: kid, k, n, mem, vst ← (∅, ∅, ∅, ∅, ∅)
4: st ← (x, kid, k, n, mem, kst, vst)
5: return st, m'
```

$\Pi_{\mathsf{SCM}}(st:\{st.x = \texttt{kex}\}, m)$

```
1: st.kst, m' ← Π_POST-GKA(st.kst, m)
2: if st.kst.x = accept then
3:    st.x, st.kid, st.k, st.n ← (vrfy, st.kst.kid, st.kst.k, st.kst.n)
4:    st.vst, m* ← Π_VRFY(⊥, (st.kid, st.n))
5:    return st, m'||m*
6: elseif st.kst.x = reject then
7:    st.x ← reject
8:    return st, m'
9: else return st, m'
```

$\Pi_{\mathsf{SCM}}(st:\{st.x = \texttt{vrfy}\}, m)$

```
1: st.vst, m' ← Π_VRFY(st.vst, m)
2: if st.vst.x = accept then
3:    st.x ← accept
4:    st.mem ← st.vst.mem
5: elseif st.vst.x = reject then
6:    st.x = reject
7: return st, m'
```

Figure 6: The SCM protocol composes a post-authenticated group key agreement protocol, $\Pi_{\mathsf{POST\text{-}GKA}}$, with a verification ceremony, Π_{VRFY}.

Theorem 4 (Compositional Authentication of SCM)**.** *For any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exists PPT adversary $\mathcal{B}$, for which*

$$\mathsf{Adv}^{\mathsf{SCM},\lambda,L,n_p,n_s}_{\mathsf{Auth}}(\mathcal{A}) \quad \leq \quad \mathsf{Adv}^{\mathsf{VRFY},n_p,n_s}_{\mathsf{Auth}}(\mathcal{B})$$

where n_p is the number of parties, n_s is the number of sessions, λ is the security parameter, L is the length of the security code, and $\mathcal{H}$ acts as the idealized human authentication functionality.

Proof. The proof consists of a singular game hop, in which we reduce the authentication of the underlying verification ceremony to that of our composed protocol. It proceeds as follows.

Game 0**:** We inline the implementation of SCM into the authentication security experiment for H2H-GKA protocols, i.e., we instantiate $\mathsf{Auth}^{\Pi,\Lambda}_{\mathsf{H2H\text{-}GKA}}(\mathcal{A})$ with $\Pi = \Pi_{\mathsf{SCM}}$. This is solely a syntactic change that does not affect the behavior of the challenger, giving $\mathsf{Adv}^{\mathsf{SCM},\Lambda}_{\mathsf{H2H\text{-}GKA}}(\mathcal{A}) = \mathsf{Adv}_{\mathsf{G0}}$.

Game 1**:** We rely on the authentication of the VRFY subprotocol to guarantee that, for every session that accepts, every honest party in the member list they output has at least one session that computed the same key context.

We introduce a new event, $\mathsf{abort}_{\mathsf{bad\text{-}vrfy}}$, that is triggered if the invariant above is broken at any point in the experiment.

Consider an adversary, $\mathcal{B}$, which we construct against a challenger, $\mathcal{C}_{\mathsf{VRFY}}$, playing an instance of $\mathsf{Auth}_{\mathsf{VRFY}}$ defined in Figure 5. Our adversary, $\mathcal{B}$, emulates Game 1 to $\mathcal{A}$ with the following changes.

First, we mirror the creation of parties directly to $\mathcal{C}_{\mathsf{VRFY}}$ and maintain a mapping of $\mathcal{C}_{\mathsf{VRFY}}$ party indices to those in the game we emulate to $\mathcal{A}$. Whenever an SCM session completes the key agreement sub-protocol, having advanced to the $x = \texttt{vrfy}$ stage, we initialize a session of the verification ceremony with its value set to the key context of the SCM session (using the $\mathcal{C}_{\mathsf{VRFY}}$ challenger's session initialization oracle, $\mathcal{O}$-Init). We then proceed to proxy messages to the appropriate $\mathcal{C}_{\mathsf{VRFY}}$ session, maintaining a mapping from $\mathcal{C}_{\mathsf{VRFY}}$ session indices to those in the game we are emulating to $\mathcal{A}$. In doing so, we forward our adversary's calls to $\mathcal{H}$.Gen and $\mathcal{H}$.Mapback to those oracles provided to us by $\mathcal{C}_{\mathsf{VRFY}}$. We return control to $\mathcal{C}_{\mathsf{VRFY}}$ when the inner adversary returns control to us.

If $\mathsf{abort}_{\mathsf{bad\text{-}vrfy}}$ is triggered during this time, we return control to $\mathcal{C}_{\mathsf{VRFY}}$ immediately, with the knowledge that the authentication invariant of the $\mathsf{Auth}_{\mathsf{VRFY}}$ game has been broken.

$$\mathsf{Adv}_{\mathsf{G0}} \leq \mathsf{Adv}^{\mathsf{VRFY},n_p,n_s}_{\mathsf{Auth}}(\mathcal{B}) + \mathsf{Adv}_{\mathsf{G1}}$$

The change introduced in Game 1 guarantees that, if any session accepts, then every honest party in its member list will have a matching honest session in the experiment. Thus, AUTH-INV always holds in Game 1 and no adversary may win the experiment with any additional advantage, completing our proof. □

Theorem 5 (Compositional Match Security of SCM)**.** *For any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exists PPT adversaries $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$, for which*

$$\mathsf{Adv}^{\mathsf{SCM},(\lambda,L,n_p,n_s)}_{\mathsf{Match}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{VRFY},n_p,n_s}_{\mathsf{Auth}}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{POST\text{-}GKA},\lambda,L,n_s}_{\mathsf{Match}}(\mathcal{C}) + \mathsf{Adv}^{\mathsf{VRFY},n_p,n_s}_{\mathsf{Match}}(\mathcal{D})$$

where H_{com} is modeled as a random oracle $\mathcal{O}$-RO_{com} (handling at most n_{com} queries), H_{sc} is modeled as a random oracle $\mathcal{O}$-RO_{sc} (handling at most n_{sc} queries), n_p is the number of parties, n_s is the number of sessions, λ is the security parameter, L is the length of the security code, and $\mathcal{H}$ acts as the idealized human authentication functionality.

Proof. We proceed with a series of games.

Game 0**:** We inline the implementation of SCM into the match security experiment for H2H-GKA protocols, i.e., we instantiate $\mathsf{Match}^{\Pi,\Lambda}_{\mathsf{H2H\text{-}GKA}}(\mathcal{A})$ with $\Pi = \Pi_{\mathsf{SCM}}$ and $\Lambda = (\lambda, L, n_p, n_s)$. This is solely a syntactic change that does not affect the behavior of the challenger, giving $\mathsf{Adv}^{\mathsf{SCM},\lambda,L,n_p,n_s}_{\mathsf{Match}}(\mathcal{A}) = \mathsf{Adv}_{\mathsf{G0}}(\mathcal{A})$.

Game 1**:** We introduce an abort event, $\mathsf{abort}_{\mathsf{spare\text{-}slot}}$, that is triggered whenever a session, π^i_s, accepts with a member list containing only honest parties, such that $\mathsf{Honest?}(i, ST[i,s].mem) = 1$, but the length of the member list is greater than the number of matching honest sessions in the experiment, $ST[i,s].n > |\mathsf{Matches}(ST[i,s].kid)|$. In other words, the abort is triggered whenever a session that satisfies the prerequisites for match security calculates a member list with a spare slot that is not accounted for by an honest session in the experiment. We bound the probability of such an event occurring with a reduction that proceeds similarly to that used in Game 1 in the proof of Theorem 4, with the following changes.

If $\mathsf{abort}_{\mathsf{spare\text{-}slot}}$ is triggered, we have that there exists a VRFY session, say π^i_s, for which there is an honest party in its member list, say $\hat{j} \in ST[i,s].mem$ such that $\mathcal{H}.\mathsf{Mapback}(i,\hat{j}) = j$ and $j \in H$, but no matching session under control of that party exists: $\nexists t : ST[j,t].v = ST[i,s].v$, where $ST[i,s]$ has been initialized with the key context of the appropriate emulated H2H-GKA session. Thus, whenever $\mathsf{abort}_{\mathsf{spare\text{-}slot}}$ is triggered, we may return control to $\mathcal{C}_{\mathsf{VRFY}}$ immediately with the knowledge that the authentication invariant of the $\mathsf{Auth}_{\mathsf{VRFY}}$ game has been broken.

$$\mathsf{Adv}_{\mathsf{G0}} \leq \mathsf{Adv}^{\mathsf{VRFY},n_p,n_s}_{\mathsf{Auth}}(\mathcal{B}) + \mathsf{Adv}_{\mathsf{G1}}$$

The changes introduced in Game 1 guarantee that, if any session has accepted, and every person in its member list is honest, then there is a corresponding session for each person in that list which is both honest (controlled by the challenger) and matching (having computed the same key context). Further, by construction, we have that each such matching session must have transitioned into the $x = \texttt{vrfy}$ stage and must have an underlying key exchange that has accepted (by lines 2–5 of the $st.x = \texttt{kex}$ case in Figure 6).

Now, consider a set of sessions that have accepted with a matching key context and for which every party identifier in every session's member list is honest (controlled by the challenger). While there may exist other matching sessions that have not accepted, we know that each of these must have an underlying key exchange session that has accepted (by the reasoning in the previous paragraph).

Game 2**:** We introduce two new abort events. The first, $\mathsf{abort}_{\mathsf{key\text{-}mismatch}}$, is triggered if any set of sessions satisfy the match security prerequisites, but do not agree on the secret key. The second, $\mathsf{abort}_{\mathsf{wrong\text{-}size}}$, is triggered if any set of sessions satisfy the match security prerequisites but do not agree on the size of the group, or have agreed upon a group size that is not equal to the number of matching sessions in the experiment. We bound the probability of either event occurring as follows.

Consider an adversary, $\mathcal{C}$, which we construct against a challenger $\mathcal{C}_{\mathsf{POST\text{-}GKA}}$ playing an instance of

$\mathsf{Match}_{\mathsf{POST\text{-}GKA}}$, the match security game for post-authenticated group key agreement defined in Figure 4. Our adversary, $\mathcal{C}$, emulates Game 2 to $\mathcal{A}$ with the following changes.

First, whenever a new instance of the POST-GKA sub-protocol is instantiated, we do so through a call to $\mathcal{C}_{\mathsf{POST\text{-}GKA}}$'s session initialization oracle, $\mathcal{O}$-Init, instead. This returns a session identifier which we take note of and associate with the appropriate party and session identifiers in our emulated experiment. We proceed to proxy all messages during the $x = \texttt{kex}$ stage of each SCM session to the appropriate $\mathcal{C}_{\mathsf{POST\text{-}GKA}}$ session. We utilize the $\mathcal{O}$-Reveal query provided by $\mathcal{C}_{\mathsf{POST\text{-}GKA}}$ to determine the true value of each session's shared key, when needed, and treat the inner adversary's queries to $\mathcal{O}$-Chall and $\mathcal{O}$-Reveal as normal. We continue until the inner adversary returns control to us, in which case we return control to $\mathcal{C}_{\mathsf{POST\text{-}GKA}}$.

If $\mathsf{abort}_{\mathsf{key\text{-}mismatch}}$ or $\mathsf{abort}_{\mathsf{wrong\text{-}size}}$ is triggered, there exists a set of matching sessions each of which has an underlying key exchange session that has accepted, and has computed a member list that is no larger than the number of matching sessions in $\mathcal{C}_{\mathsf{POST\text{-}GKA}}$'s experiment, *but* either (1) these sessions do not agree on the secret key, (2) do not agree on the correct group size, or (3) agree on a group size that is not equal to the number of matching sessions in the experiment. In other words, this set of sessions break the match security of the underlying POST-GKA sessions. Thus, whenever $\mathsf{abort}_{\mathsf{key\text{-}mismatch}}$ or $\mathsf{abort}_{\mathsf{wrong\text{-}size}}$ is triggered, we may return control to $\mathcal{C}_{\mathsf{POST\text{-}GKA}}$ immediately with the knowledge that the match invariant of the $\mathsf{Match}_{\mathsf{POST\text{-}GKA}}$ game has been broken.

$$\mathsf{Adv}_{\mathsf{G1}} \leq \mathsf{Adv}^{\mathsf{POST\text{-}GKA},\lambda,L,n_s}_{\mathsf{Match}}(\mathcal{C}) + \mathsf{Adv}_{\mathsf{G2}}$$

The introduction of $\mathsf{abort}_{\mathsf{key\text{-}mismatch}}$ directly satisfies the first requirement of the H2H-GKA match invariant, while the introduction of $\mathsf{abort}_{\mathsf{wrong\text{-}size}}$ directly satisfies the second. We proceed to guarantee the third requirement of match security, that all matching and accepting sessions compute the same member list when converted from party-local to global identifiers, using the match security of the underlying verification ceremonies, VRFY, in combination with the guarantees provided by the introduction of $\mathsf{abort}_{\mathsf{wrong\text{-}size}}$.

Game 3: We introduce an abort event, $\mathsf{abort}_{\mathsf{mem\text{-}mismatch}}$, that is triggered if there exists a set of matching and accepting sessions, for which each member list only contains the identities of honest parties, but the sessions do not agree on the member list (once the party-local identifiers are converted to global identifiers). We bound the probability of either event occurring with a reduction that proceeds similarly to that of Game 1, albeit with the following change.

Rather than tracking the $\mathsf{abort}_{\mathsf{spare\text{-}slot}}$ event, we detect instances of the $\mathsf{abort}_{\mathsf{mem\text{-}mismatch}}$ event instead. Since the member list of SCM sessions is simply a duplicate of the member list computed by the underlying VRFY session, whenever $\mathsf{abort}_{\mathsf{mem\text{-}mismatch}}$ occurs, we have that there exists a set of matching and accepting VRFY sessions, for which each member list only contains the identities of honest parties, but the sessions do not agree on the member list (once the party-local identifiers are converted to global identifiers).

Note that, by construction, an SCM session will not not accept unless the size of the member list in the underlying verification ceremony is equal to the size of the group output by the underlying key exchange and the underlying verification ceremony has accepted, itself (see lines 2–5 of the $st.x = \texttt{vrfy}$ case in Figure 6). It follows that whenever $\mathsf{abort}_{\mathsf{mem\text{-}mismatch}}$, there exist verification ceremonies in the $\mathsf{Match}_{\mathsf{VRFY}}$ game that each underlie one of the emulated SCM sessions that, together, triggered the abort condition. Together, these underlying verification ceremonies satisfy the preconditions for match security in the verification ceremony, but have calculated different member lists (when the party-local identifiers are converted to global identifiers). Thus, whenever $\mathsf{abort}_{\mathsf{mem\text{-}mismatch}}$ is triggered, we may return control to $\mathcal{C}_{\mathsf{VRFY}}$ immediately with the knowledge that the match invariant of the $\mathsf{Match}_{\mathsf{VRFY}}$ game has been broken.

$$\mathsf{Adv}_{\mathsf{G2}} \leq \mathsf{Adv}^{\mathsf{VRFY},n_p,n_s}_{\mathsf{Match}}(\mathcal{D}) + \mathsf{Adv}_{\mathsf{G3}}$$

Thanks to the changes introduced in Game 1, Game 2, and Game 3, MATCH-INV holds in the final game with certainty, completing our proof. □

Theorem 6 (Compositional Key Indistinguishability of SCM)**.** *For any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exists PPT adversaries $\mathcal{B}$ and $\mathcal{C}$, for which*

$$\mathsf{Adv}^{\mathsf{SCM},\lambda,L,n_p,n_s}_{\mathsf{KeyInd}}(\mathcal{A}) \quad \leq \\ \mathsf{Adv}^{\mathsf{VRFY},n_p,n_s}_{\mathsf{Auth}}(\mathcal{B}) \; + \; \mathsf{Adv}^{\mathsf{POST\text{-}GKA},\lambda,L,n_s}_{\mathsf{KeyInd}}(\mathcal{C})$$

where H_{com} is modeled as a random oracle $\mathcal{O}$-RO_{com} (handling at most n_{com} queries), H_{sc} is modeled as a random oracle $\mathcal{O}$-RO_{sc} (handling at most n_{sc} queries), n_p is the number of parties, n_s is the number of sessions, λ is the security parameter, L is the length of the security code, and $\mathcal{H}$ acts as the idealized human authentication functionality.

Proof. We proceed with a series of games.

Game 0: We inline the implementation of SCM into the key indistinguishability security experiment for H2H-GKA protocols, i.e., we instantiate the $\mathsf{KeyInd}^{\Pi,\Lambda}_{\mathsf{H2H\text{-}GKA}}(\mathcal{A})$ security experiment with $\Pi = \Pi_{\mathsf{SCM}}$. This is solely a syntactic change that does not affect the behavior of the challenger, giving $\mathsf{Adv}^{\mathsf{SCM},\Lambda}_{\mathsf{KeyInd}}(\mathcal{A}) = \mathsf{Adv}_{\mathsf{G0}}$.

As in our previous proofs, we rely on the fact that the verification ceremony is initialized with the key context output by the underlying key exchange protocol (see line 4 of the $st.x = \texttt{kex}$ case in Figure 6) and that the member list is duplicated directly from the underlying verification ceremony (see line 4 of the $st.x = \texttt{vrfy}$ case in Figure 6).

Game 1 proceeds identically to that of Game 1 in the proof of Theorem 5. With the changes Game 1 introduces, any session that is successfully challenged, i.e., that satisfies the prerequisites for key indistinguishability, will have an underlying key agreement session that satisfies the prerequisites for key indistinguishability in the POST-GKA experiment.

Game 2**:** We now rely on the key indistinguishability of the underlying POST-GKA sessions to guarantee the key indistinguishability of the SCM sessions.

We do so by introducing the following change. The challenger always returns a random key from $\mathcal{O}$-Chall. That is, rather than returning the real key in the $b = 0$ case of $\mathcal{O}$-Chall, the challenger uniformly samples a bit-string of length λ as the key.

Consider an adversary, $\mathcal{C}$, which we construct against a challenger $\mathcal{C}_{\mathsf{POST\text{-}GKA}}$ playing an instance of the key indistinguishability security game for post-authenticated group key agreement, as defined by $\mathsf{KeyInd}_{\mathsf{POST\text{-}GKA}}$ in Figure 4.

We construct $\mathcal{C}$ by emulating Game 2 to the adversary $\mathcal{A}$, with the following changes. Whenever a new instance of the POST-GKA sub-protocol should be instantiated, we call the $\mathcal{O}$-Init oracle provided by $\mathcal{C}_{\mathsf{POST\text{-}GKA}}$ instead. This returns a session identifier which we take note of. We proceed to proxy all messages during the $x = \texttt{kex}$ stage of each SCM session to the appropriate session in $\mathsf{KeyInd}_{\mathsf{POST\text{-}GKA}}$.

We continue until the inner adversary returns control to us with its guess b' of the challenge bit b, in which case we return control to $\mathcal{C}_{\mathsf{POST\text{-}GKA}}$ forwarding on the inner adversary's guess.

Note that, when the inner adversary makes calls to our $\mathcal{O}$-Chall and $\mathcal{O}$-Reveal oracles, we simply proxy the request as normal. Since the key context used by SCM is exactly the key context output by the underlying key exchange session, we can be sure that the *WK* and *CH* sets will track equivalently between $\mathcal{C}_{\mathsf{POST\text{-}GKA}}$ and our emulated H2H-GKA experiment. It follows that any $\mathcal{O}$-Chall or $\mathcal{O}$-Reveal query that succeeds in the H2H-GKA will succeed in the $\mathsf{KeyInd}_{\mathsf{POST\text{-}GKA}}$ game, *if* it is the case that the condition in line 4 of $\mathcal{O}$-Chall in H2H-GKA implies the condition in line 4 of $\mathcal{O}$-Chall in $\mathsf{KeyInd}_{\mathsf{POST\text{-}GKA}}$. The changes introduced in Game 1 guarantee that this is the case. It follows that our emulation is not detectable to the adversary.

Thus, we find that

$$\mathsf{Adv}_{\mathsf{G1}} \leq \mathsf{Adv}^{\mathsf{POST\text{-}GKA},\lambda,L,n_s}_{\mathsf{KeyInd}}(\mathcal{C}) + \mathsf{Adv}_{\mathsf{G2}}.$$

Since, in Game 2, the challenger behaves identically regardless of the value of the challenge bit b, the adversary has no advantage in winning Game 2: $\mathsf{Adv}_{\mathsf{G2}} = 0$. This completes our proof. □

F.2 Post-Authenticated Key Agreement

Figure 7 provides a pseudocode description of POST-GKA to accompany the prosaic description in Section 4.

F.2.1 Security Analysis

We proceed to analyze the security of this construction, finding that it provides match security, authentication, and key indistinguishability as defined in Appendix C.

Theorem 7 (Match Security of crLGKA)**.** *For any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exists PPT adversaries $\mathcal{B}$ and $\mathcal{C}$, for which*

$$\mathsf{Adv}^{\mathsf{crLGKA},\lambda,L,n_s}_{\mathsf{Match}}(\mathcal{A}) \leq \frac{(n_s)^2}{2^L} + \frac{(n_{sc})^2}{2^\lambda} + 2 \cdot \frac{(n_{com})^2}{2^\lambda} + n_s \cdot \mathsf{Adv}^{\mathsf{SIG},\lambda,n_s}_{\mathsf{EUF\text{-}CMA}}(\mathcal{B}) + n_s \cdot \mathsf{Adv}^{\mathsf{PKE},\lambda,n_s}_{\mathsf{Correct}}(\mathcal{C})$$

where H_{com} is modeled as a random oracle $\mathcal{O}$-RO_{com} (handling at most n_{com} queries), H_{sc} is modeled as a random oracle $\mathcal{O}$-RO_{sc} (handling at most n_{sc} queries), n_p is the number of parties, n_s is the number of sessions, λ is the security parameter, and L is the length of the security code.

Proof. We start by considering a series of games.

Game 0**:** We inline the implementation of crLGKA into the POST-GKA match security experiment, i.e., we instantiate $\mathsf{Match}^{\Pi,\Lambda}_{\mathsf{POST\text{-}GKA}}(\mathcal{A})$ with $\Pi = \Pi_{\mathsf{crLGKA}}$.

This is solely a syntactic change that does not affect the behavior of the challenger. It follows that the two advantages are equal, giving $\mathsf{Adv}^{\mathsf{crLGKA}}_{\mathsf{Match}}(\mathcal{A}) = \mathsf{Adv}_{\mathsf{G0}}$.

In the following set of games, we introduce a series of changes to ensure that the combined nonce (rs) is unguessable by the adversary until the point in time where all honest sessions (that contributed towards this combined nonce) have successfully processed a `create` or `join` message. To do so, we will rely on the commitment scheme to be binding and hiding.

Game 1**:** In this game, we introduce a table, COM, to store all commitments and openings that the challenger is aware of. Whenever the challenger generates a commitment on behalf of a session through a call to $\mathcal{O}$-RO_{com}, we simply save its value alongside the opening in the table. Whenever an honest session receives an adversarially generated commitment (one that was not generated by an honest session), we extract its opening from the table used to implement the

$\Pi_{\mathsf{crLGKA}}(st:\bot, m:\bot)$

1 : $x \leftarrow \mathtt{kex}$
2 : $k, kid, pks, n, coms \leftarrow \emptyset$
3 : $r \leftarrow\!\!\$\ \{0,1\}^\lambda$
4 : $ssk, svk \leftarrow\!\!\$\ \mathsf{SIG.KeyGen}(1^\lambda)$
5 : $esk, epk \leftarrow\!\!\$\ \mathsf{PKE.KeyGen}(1^\lambda)$
6 : $st \leftarrow (x, k, kid, pks, n, ssk, svk, esk, epk, r, coms)$
7 : **return** $st, (svk, epk, \mathsf{H}_{com}(r))$

$\Pi_{\mathsf{crLGKA}}(st:\{st.x = \mathtt{kex}\}, m:(\mathtt{create}, pks, coms))$

1 : $n \leftarrow |pks|$
2 : **require** $|coms| = n$
3 : **require** $pks[1:] = \mathsf{lex\text{-}sort}(pks[1:])$
4 : **require** $st.svk = pks[0]$
5 : **require** $coms[0] = \mathsf{H}_{com}(st.r)$
6 : $st.coms \leftarrow coms$
7 : $st.pks \leftarrow pks$
8 : $st.n \leftarrow n$
9 : $st.x \leftarrow \mathtt{finalize}$
10 : $st.k \leftarrow\!\!\$\ \{0,1\}^\lambda$
11 : $C \leftarrow \emptyset$
12 : **for** $epk \in pks[1:]$ **do**
13 : $\quad c \leftarrow \mathsf{PKE.Enc}(epk, st.k)$
14 : $\quad \sigma \leftarrow \mathsf{SIG.Sign}(st.ssk, c)$
15 : $\quad C \leftarrow C \cup \{(epk, c, \sigma)\}$
16 : **return** $st, (C, st.r)$

$\Pi_{\mathsf{crLGKA}}(st:\{st.x = \mathtt{kex}\}, m:(\mathtt{join}, pks, coms, c, \sigma))$

1 : $n \leftarrow |pks|$
2 : **require** $|coms| = n$
3 : **require** $pks[1:] = \mathsf{lex\text{-}sort}(pks[1:])$
4 : **let** i **st** $pks[i] = st.epk \ \wedge\ i \geq 1$
5 : **require** $i \neq \bot$
6 : **require** $coms[i] = \mathsf{H}_{com}(st.r)$
7 : $st.coms \leftarrow coms$
8 : $st.pks \leftarrow pks$
9 : $st.n \leftarrow n$
10 : $svk_I \leftarrow st.pks[0]$
11 : **require** $\mathsf{SIG.Verify}(svk_I, c, \sigma) = 1$
12 : $k \leftarrow \mathsf{PKE.Dec}(st.esk, c)$
13 : **require** $k \neq \bot$
14 : $st.x \leftarrow \mathtt{finalize}$
15 : $st.k \leftarrow k$
16 : **return** $st, st.r$

$\Pi_{\mathsf{crLGKA}}(st:\{st.x = \mathtt{finalize}\}, m:(\mathtt{open}, rs))$

1 : $correctSize \leftarrow (\ |st.coms| = |rs|\ )$
2 : $validOpenings \leftarrow (\ \forall_{i=0}^{|st.pks|-1}\ :\ st.coms[i] = \mathsf{H}_{com}(rs[i])\ )$
3 : **if** $correctSize \ \wedge\ validOpenings$ **then**
4 : $\quad st.x \leftarrow \mathtt{accept}$
5 : $\quad st.kid \leftarrow \mathsf{H}_{sc}(st.pks \parallel rs)$
6 : **else**
7 : $\quad st.x \leftarrow \mathtt{reject}$
8 : **return** $st, \emptyset$

Figure 7: A post-authenticated group key agreement protocol, $\Pi_{\mathsf{POST\text{-}GKA}}$.

random oracle. The challenger saves this commitment and its opening to the *COM* table.

This change introduces extra tracking within the challenger without effecting its observable behavior. It follows that the two advantages are equal, giving $\mathsf{Adv}_{\mathsf{G0}} = \mathsf{Adv}_{\mathsf{G1}}$.

Game 2**:** In this game, we introduce an abort event, $\mathsf{abort}_{\mathsf{com\text{-}coll}}$, which is triggered if an honest session successfully processes an opening to a commitment that does not match the opening stored in our commitment table.

We can bound the probability of this event occurring by considering the implementation of H_{com} as the O-RO_{com} random oracle. To check a claimed opening of a commitment, an honest session fetches the value from the random oracle's table using the given opening as the label. It then compares the value in the table with the value of the commitment it is checking. Thus, any time an opening succeeds despite not being the expected opening stored in *COM*, we can be sure that there are two entries in the random oracle's table pointing to the same value. Since each value is selected independently and uniformly at random from the space $\{0,1\}^\lambda$, we can bound the probability of this occurring as $(n_{com})^2/2^\lambda$. Thus, we find:

$$\begin{aligned}\mathsf{Adv}_{\mathsf{G1}} &\leq \mathsf{Adv}_{\mathsf{G2}} + \Pr[\mathsf{abort}_{\mathsf{com\text{-}coll}}] \\ &\leq \mathsf{Adv}_{\mathsf{G2}} + \frac{(n_{com})^2}{2^\lambda}\end{aligned}$$

We can now be sure that, in the games that follow, commitments will only open to the value they were created with (as stored in our set *COM*).

Game 3**:** In this game, we replace the generation of every honest commitment with a random sample from the output space of the random oracle. We then delay the generation of the respective nonce until the point in time where the honest session reveals it.

This moment arises in the final line of Π_{crLGKA} in the cases where it is processing a `create` or `join` message. When it does, we sample a random nonce from the input space of the random oracle (as the opening) and place our previously generated commitment in the appropriate entry in the table. We replace the check of the client's commitment contribution against a hash of its nonce (in line 5 of the `create` case and line 6 of the `join` case) with an equality check against the commitment value itself. This change is not observable to the adversary, unless the adversary has already queried this particular value to the random oracle: we would need to either change the value for this query or retroactively change the value of our commitment, both of which are detectable by the adversary. Instead, we introduce an abort event, $\mathsf{abort}_{\mathsf{nonce\text{-}coll}}$, which is triggered whenever the nonce we generate already has an entry in the table that implements O-RO_{com}.

Since we generate this value uniformly from the random oracle's input space, and do so independently to the prior execution of this experiment, we find that the probability that any nonce collides with an existing entry in the table is at most $(n_{com})^2/2^{\lambda}$. Thus, we find:

$$\begin{aligned}\mathsf{Adv}_{\mathsf{G2}} &\leq \mathsf{Adv}_{\mathsf{G3}} + \Pr[\mathsf{abort}_{\mathsf{nonce\text{-}coll}}] \\ &\leq \mathsf{Adv}_{\mathsf{G3}} + \frac{(n_{com})^2}{2^{\lambda}}.\end{aligned}$$

As a result of this change, we can be sure that the adversary knows nothing about each honest session's nonce until it is explicitly revealed. We observe that the construction we are studying will only perform such a reveal when responding to a successfully processed group `create` or `join` message.

Game 4: We now look to demonstrate that sessions which compute the same key context also compute the same participant key list (and vice versa, that those which compute the same participant key list compute the same key context). To do so, we introduce an abort event for each, and bound the probability of such events occurring.

We start by augmenting our *COM* table to additionally mark whether the opening of any particular commitment is known to the adversary. In the case of honestly generated commitments, we set this flag to false initially, and mark it as true at the point of reveal (when a session successfully processes a `create` or `join` message). In the case of adversarially generated commitments, this flag is set to true as soon as the challenger becomes aware of the commitment (when such a commitment is included in a `create` or `join` message). Note that this change alone does not modify the observable behavior of the challenger.

Consider the challenger as they process a group creation message with commitment list *coms*, for which they are about to reveal a nonce that opens the last of those commitments (i.e., where all other commitments within *coms* are already marked as revealed in *COM*). In such cases, we have the challenger calculate the resulting key context, *kid*, directly after sampling the final nonce (right before revealing the nonce to the adversary). We do so by querying $O\text{-}\mathsf{RO}_{sc}$ with the appropriate input, and introduce the following two abort events.

First, if there already exists an entry for the label '$pks\|rs$' in the table that implements $O\text{-}\mathsf{RO}_{sc}$, then we trigger the abort event $\mathsf{abort}_{\mathsf{sc\text{-}in\text{-}coll}}$. By construction of this game hop, we can be sure that the adversary has knowledge of every component in *pks* and *rs*, *except* the last nonce. We denote this final nonce r, and note that it is sampled during processing of the current query. Its value is, thus, distributed uniformly at random and is independent of the protocol execution: the adversary has no information about its content before this query is made. Note that it is not possible for two honest sessions to generate the same nonce, since this is exactly the condition of the abort event introduced in Game 3.

Over the course of the experiment, the adversary is able to make up to n_{sc} queries to the $O\text{-}\mathsf{RO}_{sc}$ oracle. If at any point in time, the $\mathsf{abort}_{\mathsf{sc\text{-}in\text{-}coll}}$ event is triggered, then the challenger has generated a nonce r uniformly at random and independently of prior protocol execution, but the adversary has already executed the query $O\text{-}\mathsf{RO}_{sc}(pks\|rs)$, for which $r \in rs$. Thus, the probability of such an event occurring is at most $(n_{sc})^2/2^{\lambda}$.

Second, we trigger the abort event $\mathsf{abort}_{\mathsf{sc\text{-}out\text{-}coll}}$ if this random oracle query results in a key context that has already been computed for a session *whose participant key list does not match*. Since the two sessions have differing participant key lists, we can be sure that their inputs into the random oracle were different. Thus, each of these key contexts were sampled uniformly at random and independent of the others, leading to the probability of such a collision being at most $(n_s)^2/2^L$. Thus, we find:

$$\begin{aligned}\mathsf{Adv}_{\mathsf{G3}} &\leq \mathsf{Adv}_{\mathsf{G4}} + \Pr[\mathsf{abort}_{\mathsf{sc\text{-}in\text{-}coll}}] + \Pr[\mathsf{abort}_{\mathsf{sc\text{-}out\text{-}coll}}] \\ &\leq \mathsf{Adv}_{\mathsf{G4}} + \frac{(n_{sc})^2}{2^{\lambda}} + \frac{(n_s)^2}{2^L}\end{aligned}$$

The introduction of $\mathsf{abort}_{\mathsf{sc\text{-}in\text{-}coll}}$ guarantees that honest sessions which provide the same input are guaranteed to compute the same key context. On the other hand, the introduction of $\mathsf{abort}_{\mathsf{sc\text{-}out\text{-}coll}}$ guarantees that any honest sessions which compute the same key context are guaranteed to have provided the same input to do so. Thus, from Game 4 onwards, all sessions with matching key contexts also have a matching view of the participant key list (and vice versa). Since the participant key list encodes the group size (calculated as the length of the participant key list) and the cryptographic identity of the leader (at the first position in the list), we can be sure that matching sessions agree on these, too.

In the next set of games, we introduce changes to guarantee that the distribution of the meeting key between *honest* sessions is done so without tampering (relying on the unforgeability of the signature scheme) and correctly (relying on the correctness of the public-key encryption scheme).

Game 5: We rely on the existential unforgeability of the SIG signature scheme to guarantee that honest sessions will only accept key distribution ciphertexts that were created using the signing key of the leader, providing that the respective verification key was generated by an honest session.

Whenever an honest session, say π_s, generates a signature σ over some message m, the challenger saves the message m into a table addressed by π_s's verification key $ST[s].svk$. We introduce an abort event $\mathsf{abort}_{\mathsf{sig\text{-}forge}}$ that is triggered whenever an honest session successfully verifies a message and signature pair, e.g., (m', σ'), as originating from

$ST[s].svk$ but the message m' is not in the aforementioned entry of the table.

We bound the probability of $\mathsf{abort}_{\mathsf{sig\text{-}forge}}$ occurring through a series of hybrids: Game 4.0, Game 4.1, ..., Game 4.$\mathsf{n_s}$. The initial hybrid, Game 4.0, is exactly Game 4 and the final hybrid, Game 4.$\mathsf{n_s}$, is exactly Game 5.

The h-th hybrid, Game 4.h, works as in Game 5 but only tracks the messages signed by the first h sessions initialized in the experiment. We now bound the difference in advantage of any adversary between two consecutive hybrids: Game 4.h and Game 4.h + 1.

Consider an adversary, $\mathcal{D}$, which we construct against a challenger $\mathcal{C}_{\mathsf{EUF\text{-}CMA}}$ playing the EUF-CMA game for the existential unforgeability of a signature scheme SIG [30, Def 13.1]. We do this by emulating Game 4.h + 1 to an adversary, $\mathcal{A}$, with the following changes to Game 4.h + 1. Let π_{h+1} denote the $(h+1)$-th session initialized in the security experiment. We replace π_{h+1}'s signing key pair with the verification key supplied by $\mathcal{C}_{\mathsf{EUF\text{-}CMA}}$. When π_{h+1} needs to sign a message, we replace calls to the signing function SIG.Sign with queries to $\mathcal{C}_{\mathsf{EUF\text{-}CMA}}$'s signing oracle. Keeping a table of queried messages, $\mathcal{D}$ waits for a session to receive a message-signature pair whose message was not queried to the signing oracle but which verifies correctly. If one occurs, $\mathcal{D}$ returns control to $\mathcal{C}_{\mathsf{EUF\text{-}CMA}}$ with the forgery.

Game 4.h + 1 essentially splits the event space of Game 4.h into two, cases where $\mathcal{A}$ successfully forges a signature for $ST[h+1].svk$ and cases where it does not. The reduction above simply guarantees that we can convert any of these forgeries into a valid break of the EUF-CMA game. Thus, we may bound the probability of any adversary winning Game 4.h with $\mathsf{Adv}_{\mathsf{G4.h}} \leq \mathsf{Adv}^{\mathsf{SIG},\lambda,n_s}_{\mathsf{EUF\text{-}CMA}}(\mathcal{D}) + \mathsf{Adv}_{\mathsf{G4.h+1}}$.

Iteratively applying the bounds we have derived for each of our hybrids, and recalling that Game 4.0 is Game 4 and Game 4.$\mathsf{n_s}$ is Game 5, gives

$$\mathsf{Adv}_{\mathsf{G4}} \leq n_s \cdot \mathsf{Adv}^{\mathsf{SIG},\lambda,n_s}_{\mathsf{EUF\text{-}CMA}}(\mathcal{B}) + \mathsf{Adv}_{\mathsf{G5}}.$$

Game 6**:** We rely on the correctness of the PKE public-key encryption scheme to ensure that key distribution ciphertexts sent from honest leader sessions are decrypted to the intended key.

Whenever an honest session, say π_s, generates a PKE ciphertext c encrypting some plaintext message m for a recipient session π_r, the challenger saves this plaintext into a table addressed by the recipient's public encryption key epk and c. We introduce an abort event $\mathsf{abort}_{\mathsf{bad\text{-}dec}}$ that is triggered whenever an honest session decrypts a ciphertext with the secret counterpart of epk to receive a plaintext m' but it is not in the entry of the table addressed by (epk, c), *providing* that the recipient session has an honestly generated leader verification key stored in $ST[r].svk_l$.

We bound the probability of $\mathsf{abort}_{\mathsf{bad\text{-}dec}}$ occurring through a series of hybrids: Game 5.0, Game 5.1, ..., Game 5.$\mathsf{n_s}$. The initial hybrid, Game 5.0, is exactly Game 5 and the final hybrid, Game 5.$\mathsf{n_s}$, is exactly Game 6.

The h-th hybrid, Game 5.h, works as in Game 6 but only tracks the ciphertexts generated by the first h sessions initialized in the experiment. We now bound the difference in advantage of any adversary between two consecutive hybrids: Game 5.h and Game 5.h + 1.

Consider an adversary, $\mathcal{C}$, which we construct against a challenger $\mathcal{C}_{\mathsf{PKE}}$ playing the implicit correctness game for public-key encryption schemes as defined by [30, Def 12.1]. We do this by emulating Game 5.h + 1 to the adversary, $\mathcal{A}$, with the following changes to Game 5.h + 1. Let π_{h+1} denote the $(h+1)$-th session initialized in the security experiment. We replace π_{h+1}'s public encryption key pair with the public key supplied by $\mathcal{C}_{\mathsf{PKE}}$. When another honest session encrypts a message to the public key generated by π_{h+1}, we additionally save the plaintext and ciphertext to a table. Whenever π_{h+1} decrypts a ciphertext, we check against this table. If an entry for the ciphertext exists, and the session currently has an honestly generated verification key stored in $ST[r].svk_l$, but it decrypts to a different value, the resulting plaintext-ciphertext pair represents a break in the correctness of the scheme. Thus, $\mathcal{D}$ may submit the resulting plaintext-ciphertext pair to $\mathcal{C}_{\mathsf{PKE}}$.

Note that, thanks to the changes in Game 5, we can be sure that sessions with honestly generated leader verification keys stored in the $.svk_l$ field will only attempt to decrypt PKE ciphertexts that have been generated by the honest session that generated this verification key. It follows that we may bound the probability of any adversary winning Game 5.h with $\mathsf{Adv}_{\mathsf{G5.h}} \leq \mathsf{Adv}^{\mathsf{PKE},\lambda,n_s}_{\mathsf{Correct}}(\mathcal{C}) + \mathsf{Adv}_{\mathsf{G5.h+1}}$.

Iteratively applying the bounds we have derived for each of our hybrids, and recalling that Game 5.0 is Game 5 and Game 5.$\mathsf{n_s}$ is Game 6, gives

$$\mathsf{Adv}_{\mathsf{G5}} \leq n_s \cdot \mathsf{Adv}^{\mathsf{PKE},\lambda,n_s}_{\mathsf{Correct}}(\mathcal{C}) + \mathsf{Adv}_{\mathsf{G6}}.$$

We now determine the probability of the adversary winning the final game, Game 6.

The changes introduced in Game 1 to Game 4 guarantee that any group of honest sessions will compute a matching key context *if and only if* they have computed the same participant key list and commitment list. By the deterministic structure of the participant key list, it follows that they agree on the size of the group and the verification key of the leader.

Let $\{(s_x, st_x)\}_{x=0}^{n-1}$ denote one such group of honest sessions, where v is the agreed upon key context such that $\{(s_x, st_x)\}_{x=0}^{n-1} = \mathsf{Matches}(v)$. In what follows, we additionally ask that there exists at least one matching session

that has computed a group size no larger than the number of honest matching sessions in the experiment, n. In other words, we ask this group of sessions to satisfy the preconditions for match security (as defined by MATCH-INV in Figure 4).

Note that a session will only accept a `create` (or `join`) message if its verification key (or public encryption key) is (a) present in the list and (b) occupies the same position in the participant key list that its commitment occupies in the commitment list. Condition (a) ensures that there can be no more matching sessions in the experiment than the agreed upon group size. Taken with the existing guarantee that $(st_0.n = \ldots = st_{n-1}.n)$ and $(\exists_{x=0}^{n-1} : st_x \leq n)$, we have that $(st_0.n = \ldots = st_{n-1}.n = n)$.

Next, in addition to its primary purpose (of preventing a commitment from opening to multiple values), the change introduced in Game 2 guarantees that no two honest sessions will produce the same commitment: the commitment output by an honest session uniquely identifies it. Combined with condition (b) from the previous paragraph, we see that every key present in the agreed upon participant key list must therefore have been generated by an honest session that computed the same key context. In other words, we can be sure that sessions in this group are communicating using honestly generated public encryption and verification keys only (their keys were faithfully distributed).

This enables us to apply the changes introduced in Game 5 and Game 6 to the key distribution messages. Game 5 guarantees that only key distribution messages sent by the leader will be accepted by the non-leading sessions. Similarly, Game 6 guarantees that such key distribution messages will always decrypt to the value encrypted by the leader. Thus, we find that $(k_0 = k_1 = \ldots = k_{n-1})$,[19] completing our proof. □

Theorem 8 (Key Indistinguishability of crLGKA)**.** *For any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exists PPT adversaries $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$, for which*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{crLGKA},\lambda,L,n_s}_{\mathsf{KeyInd}}(\mathcal{A}) \leq\ & \frac{(n_s)^2}{2^L} + \frac{(n_{sc})^2}{2^\lambda} + 2\cdot\frac{(n_{com})^2}{2^\lambda} \\ & + n_s\cdot\mathsf{Adv}^{\mathsf{SIG},(\lambda,n_s)}_{\mathsf{EUF\text{-}CMA}}(\mathcal{B}) + n_s\cdot\mathsf{Adv}^{\mathsf{PKE},(\lambda,n_s)}_{\mathsf{Correct}}(\mathcal{C}) \\ & + n_s\cdot\mathsf{Adv}^{\mathsf{PKE},(\lambda,n_s)}_{\mathsf{IND\text{-}CCA}}(\mathcal{D})\end{aligned}$$

where H_{com} is modeled as a random oracle O-RO_{com} (handling at most n_{com} queries), H_{sc} is modeled as a random oracle O-RO_{sc} (handling at most n_{sc} queries), n_p is the number of parties, n_s is the number of sessions, λ is the security parameter, and L is the length of the security code.

Proof. We start by considering a series of games.

[19] Note that our construction disallows the leader (or any session) from accepting more than one `create` or `join` message. Explicitly, acceptance of such a message puts a session in a state where it will never accept another group creation message.

Game 0**:** We inline the implementation of crLGKA into the key indistinguishability security experiment for POST-GKA protocols, i.e., we instantiate $\mathsf{KeyInd}^{\Pi,\Lambda}_{\mathsf{POST\text{-}GKA}}(\mathcal{A})$ with $\Pi = \Pi_{\mathsf{crLGKA}}$. Thus, we find that $\mathsf{Adv}^{\mathsf{POST\text{-}GKA},\Lambda}_{\mathsf{KeyInd}}(\mathcal{A}) = \mathsf{Adv}_{\mathsf{G0}}$.

Game 1 through Game 6 of this proof are defined to be the same six game hops as in the proof of Theorem 7; the only change is that the winning condition is now **if** $b' = b$ **then return** 1 (and **return** 0 otherwise) rather than breaking the match invariant; the bounds carry over unchanged.

In what follows, we use $\pi^\star$ as a placeholder for any session that meets the prerequisites for a challenge query. That is, it represents any session that has accepted with a group size for which there are at least as many matching sessions in the security experiment and, additionally, its key context is not present in the *CH* and *WK* sets at the time of the challenge query.

The changes introduced in Game 1 to Game 4 ensure that any honest sessions will compute a matching key context to $\pi^\star$ *if and only if* they have computed the same participant key and commitment lists.

The changes in Game 2 and the checks employed by clients while processing `create` and `join` messages guarantee that the agreed upon group size is correct and that every key present in the participant key list was honestly generated by a matching honest session in the experiment. In other words, we can be sure that sessions in this group are communicating using honestly generated public encryption and verification keys only.

Next, the changes introduced in Game 5 and Game 6 guarantee that the distribution of the meeting key between the group leader and following sessions (of which $\pi^\star$ may be either) is done so without tampering (relying on the unforgeability of the signature scheme) and correctly (relying on the correctness of the public-key encryption scheme).

We now look to guarantee that the contents of these key distribution messages remain secret from the adversary.

Game 7**:** We rely on the chosen-ciphertext security of the PKE public-key encryption scheme to guarantee that the key distribution ciphertexts do not leak information about the key to the adversary.

Whenever an honest leader, say π_t, generates a key k, we use the PKE scheme to encrypt the all-zero string and proceed to save k in a table addressed by the resulting ciphertext and the public-encryption key $ST[s].epk$ of the intended recipient, whom we denote π_s. We use the PKE scheme as normal whenever an intended recipient is not an honestly generated public key (those that are not in the set *PKS*), encrypting the real key k rather than an all-zero replacement. At decryption time, we check if the ciphertext is present in the table and, if so, fetch the key from our table rather than enacting the decryption. If no such ciphertext exists in the table, we proceed with decryption as normal.

We bound the probability of the adversary distinguishing Game 7 from Game 6 through a series of hybrids: Game 6.0, Game 6.1, ..., Game 6.n_s. The initial hybrid, Game 6.0, is exactly Game 6 and the final hybrid, Game 6.n_s, is exactly Game 7.

The h-th hybrid, Game 6.h, works as in Game 7 but only replaces the key distribution ciphertexts sent to the first h sessions in the game. We now aim to bound the difference in advantage of any adversary in Game 6.h compared to Game 6.h + 1.

Consider an adversary, $\mathcal{D}$, which we construct against a challenger $\mathcal{C}_{\text{IND-CCA}}$ playing the IND-CCA game [30, Def 12.8] for the chosen-ciphertext security of a PKE scheme [30, Def 12.8]. We do this by emulating Game 6.h + 1 to the adversary $\mathcal{A}$, with the following changes to Game 6.h + 1. At the beginning of the IND-CCA game, the challenger provides us with the challenge public key pk_{h+1}. Let π_{h+1} be the $(h+1)$-th session initialized in the game. Rather than generating π_{h+1}'s key pair using the PKE scheme, we use the public key provided by $\mathcal{D}$.

Whenever a leader session attempts to encrypt a meeting key for epk_{h+1}, and all keys within the leader session's participant key list were generated by honest sessions, we replace this with a call to the challenge oracle of the IND-CCA game with m_0 set to the real key k and m_1 set to the all-zero string. We additionally store the real key in the aforementioned table, addressed by the resulting ciphertext and epk_{h+1}. When decrypting ciphertexts, π_{h+1} first checks if the ciphertext is present in this table and, if so, uses it directly. If not, it queries the decryption oracle provided by the $\mathcal{C}_{\text{IND-CCA}}$ challenger. When the adversary $\mathcal{A}$ returns their guess b', we return the same guess to the $\mathcal{C}_{\text{IND-CCA}}$ challenger.

We first consider the case where neither π_{h+1} nor any matching session is successfully challenged at some point during the experiment. In this case, the only change in behavior between Game 6.h and Game 6.h + 1 that may be observable to the adversary is the replacement of honest key distribution ciphertexts to π_{h+1} with the output of $\mathcal{C}_{\text{IND-CCA}}$'s challenge encryption oracle.

Next, consider the case where either π_{h+1} or another honest session with a matching key context, is successfully challenged at some point in the experiment. Let $\pi^\star$ be the matching session that is challenged. If the challenge session is acting as the leader, we can be sure that the adversary was not able to influence the meeting key.[20]

Since the challenge is successful, we may utilize the previously determined guarantees for successfully challenged sessions to determine the following to be true at the point in time the challenge is made.

1) All such matching sessions agree upon the leader verification key and this key was generated by one of these matching honest sessions.
2) The meeting key received by following sessions is the key generated by the leading session.

Thus, if the challenge session is acting as a follower, we can also be sure that the adversary was not able to influence the meeting key.

Additionally, none of these matching sessions were the subject of a successful $\mathcal{O}$-Reveal query, and none but the challenge session were the subject of a successful $\mathcal{O}$-Chall query. This follows from the challenger's tracking of the CH and WK sets. Further still, since the challenge was successful, we can be sure that the leader session's participant key list, at the time of generating the key distribution messages, only contained honestly generated public encryption keys. It follows that the adversary is not able to use a different $\mathcal{O}$-Chall query, a $\mathcal{O}$-Reveal query, or an adversarially-controlled key pair to gain access to the real key directly.

Together, we see that the only change in behavior between Game 6.h and Game 6.h + 1 that may be observable to the adversary is the use of $\mathcal{C}_{\text{IND-CCA}}$'s challenge encryption oracle to encrypt key distribution messages to π_{h+1}.

Thus, when $\mathcal{D}$'s challenge bit is 0, we have that our game is exactly Game 6.h and, when $\mathcal{D}$'s challenge bit is 1, we have that our game is exactly Game 6.h + 1. We may therefore bound the increase in advantage of $\mathcal{A}$ between Game 6.h and Game 6.h + 1 by the advantage of any adversary in winning the IND-CCA game for the PKE scheme. Thus, we have that $\mathsf{Adv}_{\text{G6.h}} \leq \mathsf{Adv}^{\mathsf{PKE},\lambda,n_s}_{\text{IND-CCA}}(\mathcal{D}) + \mathsf{Adv}_{\text{G6.h+1}}$.

Iteratively applying the bounds we have derived from each of our hybrids, and recalling that Game 6.0 is Game 6 and Game 6.n_s is Game 7, gives $\mathsf{Adv}_{\text{G6}} \leq n_s \cdot \mathsf{Adv}^{\mathsf{PKE},\lambda,n_s}_{\text{IND-CCA}}(\mathcal{D}) + \mathsf{Adv}_{\text{G7}}$

Note that, as of Game 7, the challenger now behaves with an identical distribution regardless of the value of the challenge bit b, and so it is not possible for the adversary to distinguish between the two cases with any advantage: $\mathsf{Adv}_{\text{G7}} = 0$. It follows that the probability of the adversary winning Game 7 is zero, completing our proof. □

F.3 Two-Way Verification Ceremony

Figure 8 provides a pseudocode description of the two-way verification ceremony, 2W-VRFY, to accompany its prosaic description in Section 4.

F.3.1 Security Analysis

We now make a few observations about the behavior of this construction, which aid in the security analysis that follows.

[20] The key is sampled as a uniformly random bit string by the leader.

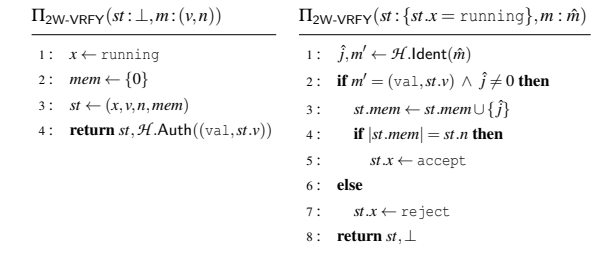

Figure 8: Two-Way Verification Ceremony, $\Pi_{\text{2W-VRFY}}$.

Observation 1. When a session s for a particular party i, π_s^i, is initialized with a message (v,n), then the state variables $st.v$ and $st.n$ are set to v and n, respectively. These values are never updated through the lifetime of the session.

Observation 2. If a session π_s^i adds a party identifier, $\hat{j} \neq i$, to its member set, so that $\hat{j} \in st.mem$, then π_s^i must have invoked $\mathcal{H}.\mathsf{Ident}(\hat{m}) \mapsto (\hat{j}, m)$ for which $m = (\mathtt{val}, st.v)$, i.e., it has received a human-authenticated message from the party locally identified as $\hat{j}$ containing the initial value, $st.v$.

Observation 3. Any honest session π_s^i will only invoke $\mathcal{H}.\mathsf{Auth}(m)$ with the value $m = (\mathtt{val}, st.v)$.

Theorem 9 (Authentication of 2W-VRFY)**.** *The* 2W-VRFY *group-value verification ceremony provides authentication with certainty, under the assumption that $\mathcal{H}$ acts as the idealized human authentication functionality.*

Proof. We consider the game that results from inlining the pseudocode definition of $\Pi_{\text{2W-VRFY}}$ into the authentication security experiment for group-value verification ceremonies, which we denote as $\mathsf{Auth}^{\text{2W-VRFY}}_{\text{VRFY}}$.

Suppose that a session, π_s^i, has the state $(\mathtt{accept}, v, mem)$. By *Observation 1*, the value v is exactly the value the session was initialized with, and this matches the value stored in $ST[i,s].v$.

If $|mem| = 1$, then π_s^i is an honest session and $ST[i,s].v = v$. If $|mem| \geq 2$, then consider a party $\hat{j}$ in mem, for which $\hat{j} \neq 0$. By *Observation 2*, this party's presence in the membership list requires that there exists a message $\hat{m}_j$, for which $\mathcal{H}.\mathsf{Ident}_i(\hat{m}_j) \mapsto \hat{j}, (\mathtt{val}, v)$.

Unforgeability of the human authentication functionality guarantees that the party with global identifier $j = \mathcal{H}.\mathsf{Mapback}(i, \hat{j})$ must have made a call to the $\mathcal{H}.\mathsf{Auth}$ oracle with $(\mathtt{val}, v)$ as input.

Note that if party j is under the control of the adversary ($j \in B$), then the authentication invariant requires nothing from this entry in mem. On the other hand, if party j is honest ($j \in H$), we must demonstrate that there exists an honest session initialized with the matching value v.

Since j is honest, we may apply *Observation 3* to see that there must exist an honest session π_t^j, for some $t \in [n_s]$, for which $ST[j,t].v = v$. Applying *Observation 1* to this session, we see that π_t^j must also have been initialized with a matching value v.

Note that, by construction, our protocol ensures that any session π_s^i contains itself as a member, i.e., $0 \in st_x.mem$ where $\mathcal{H}.\mathsf{Mapback}(i_x, 0)$ is defined to be i_x. In this case, we have that π_s^i is an honest session for which $ST[i,s].v = v$.

It follows that the authentication variant is satisfied with certainty, and the adversary is not able to win the security experiment. □

Theorem 10 (Match Security of 2W-VRFY)**.** *The* 2W-VRFY *group-value verification ceremony provides match security with certainty, under the assumption that $\mathcal{H}$ acts as an idealized human authentication functionality.*

Proof. We consider the game that results from inlining the pseudocode definition of $\Pi_{\text{2W-VRFY}}$ into the match security experiment for group-value verification ceremonies, which we denote as $\mathsf{Match}^{\text{2W-VRFY}}_{\text{VRFY}}$.

The adversary is able to win the experiment if and only if they are able to place the experiment in a state where the match security invariant, MATCH-INV, is not satisfied. We will argue why this not possible in our protocol.

Consider $\{(i_x, s_x, st_x)\}_{x=0}^{m-1} = \mathsf{Accepts}(v)$ for which $n = |\mathsf{Matches}(v)|$ for some value v. Note that, by construction, the experiment ensures that $n \geq m$, while the protocol ensures that each $st_x.mem$ contains the party i_x. Moreover, an accepting session of $\Pi_{\text{2W-VRFY}}$ satisfies $|st_x.mem| = st_x.n$ by construction; combined with the MATCH-INV precondition $st_0.n = \ldots = st_{m-1}.n = n$, we obtain $|st_x.mem| = n$ for all x. Further, if a member list $st_x.mem$ consists solely of honest parties then, by Theorem 9, we have that every party in its member list $st_x.mem$ has a matching honest session initialized with the same value v.

Each member list $st_x.mem$ consists solely of honest parties and each $|st_x.mem| = n = |\mathsf{Matches}(v)|$, which implies $\mathcal{H}.\mathsf{Mapback}(i_0, mem_0) = \mathcal{H}.\mathsf{Mapback}(i_1, mem_1) = \ldots = \mathcal{H}.\mathsf{Mapback}(i_{n-1}, mem_{n-1})$. This concludes the proof. □

Perfect authentication? Curiously, the group-value verification ceremony is secure against an information theoretic adversary, no matter the parameters used to instantiate the protocol, or its usage. This is reflected directly in the authentication guarantee of the composed protocol. To see why, notice that session partnering is defined in our model directly by the key context, i.e., a group of sessions are considered to be part of the same group exactly when they have computed matching key contexts. Since our construction defines the key context exactly as the security code that users compare, and the idealized human authentication functionality provides perfect identification of a message speaker, this result is somewhat natural: if a user believes another (honest) person to be part of

their group, they are guaranteed to be so. On the other hand, if a group of sessions believe themselves to be part of the same group, but have computed differing keys or member lists, this would be captured as a break in match security, instead.

G Example Encode and Decode Algorithms

Our first construction ties a random token to each authenticated message. The unpredictability provided by this encoding scheme enables the ideal functionality to effectively emulate a human authentication scheme within the bounds of a security experiment.

$\mathsf{Encode}(i,m)$	$\mathsf{Decode}(\hat{m}:(\mathtt{hauth},i,m,r))$
$r \leftarrow\!\$\ \{0,1\}^{\lvert m\rvert}$	**return** i,m
return $(\mathtt{hauth},i,m,r)$	

Our second construction utilizes the unforgeability of a deterministic signature scheme rather than unpredictability to, in tandem with the ideal functionality, provide the requisite properties. Let $\mathsf{dtSIG} = (\mathsf{KeyGen},\mathsf{Sign},\mathsf{Verify})$ denote a deterministic digital signature scheme. Here, the adversary generates a constant key pair $(sk,vk) \leftarrow\!\$\ \mathsf{dtSIG.KeyGen}(1^\lambda)$ before providing the following two algorithms to the challenger.

$\mathsf{Encode}(i,m)$	$\mathsf{Decode}(\hat{m}:(\mathtt{hauth},i,m,\sigma))$
$\sigma \leftarrow \mathsf{dtSIG.Sign}(sk,(\mathtt{hauth},i,m))$	**require** $\mathsf{dtSIG.Verify}(vk,(\mathtt{hauth},i,m),\sigma)$
return $(\mathtt{hauth},i,m,\sigma)$	**return** i,m

H Further Details on Prior Work

Implementations. Microsoft Teams shows meeting participants a security code and instructs participants to verify that they see the same code [42]; however, it is not fully clear whether this is intended to be done in-band. A simple test confirms that the code changes when any participant joins or leaves the meeting, requiring a repeat of the verification ceremony. Neither the details of the security code protocol (including how the code is computed) nor a formal security analysis are available in public documentation.

Zoom provides a fundamentally different implementation of the security code method compared to Teams; in Zoom the code is tied to a specific participant (a meeting leader) [10,61]. This comes with the caveat that all parties must be aware of who the leader is and the special role they play. This is captured within the verification ceremony, whereby the leader must always speak first.[21] As the code does not encode the set of participants, Zoom adds a liveness check (a "heartbeat") that continually distributes this information to the participants from the leader's view, ensuring everyone has a consistent view of the meeting. Zoom provides many details of their system in [10], including stating that verification requires each participant to read out the security code and that the leader needs to go first. They also clearly state that the code can be read on the in-band channel; however, no formal security analysis is presented. Note that The Zoom protocol was studied in [22], but this analysis leaves out security codes as out-of-scope, instead building on the assumption that a trusted PKI can be used to verify the keys; such an assumption is incompatible with the definition of E2EE that we use. As a protection against deepfakes, [10, Section 7.7] suggests out-of-band verification. In contrast, we provide a security code method with clear security guarantees (Figure 2) without requiring any participant to behave differently from others and without requiring a continual liveness check.

Cisco Webex prefers Single Sign-On (SSO) to verify meeting participants' identities [20], which does not yield true E2EE guarantees. For non-SSO participants, they support a security code mechanism, where the code is derived from the MLS epoch authenticator [6]. In the Webex implementation, the code changes every time someone joins the meeting, but not when someone leaves. While [6] clearly states that the MLS epoch authenticator codes need to be verified out-of-band, the Webex whitepaper [20] is not clear about this. While security analysis of MLS exists [3,5,14,58], there is no analysis that would include in-band verification of a security code derived from the epoch authenticator.

Signal, WhatsApp,[22] and Discord explicitly recommend out-of-band verification [21, 52, 60], although Signal documentation [52] seems to suggest that in-band verification is possible.

Ceremony analysis. In 2007, Ellison [26] introduced *ceremony design and analysis*, defining ceremonies as protocols involving both human nodes and non-traditional communication channels. While the analysis in [26] was in a symbolic Dolev-Yao model [23], recent works on Bluetooth passkey entry ceremonies [54] and user-mediated authentication (e.g., key fingerprint comparison in Signal) [24] are in the computational model and more closely relate to our work. However, these two papers operate in a peer-to-peer out-of-band setting, whereas we focus on in-band verification in a multi-participant setting.

Key exchange. We capture the security of our construction through a group key exchange formalism. Initial development of models for two-party key exchange [7, 9, 18] was followed soon by generalizations to the group setting [11–13]. Manulis [40] provided a survey of group key exchange models as of 2006. More recently, [48] presents a systematization of knowledge on game-based models for group key exchange alongside a proposal for such a model; the formalism we introduce follows closely from this approach.

[21] Thus, while Zoom claims that this approach brings usability benefits [10, Section 7.7], it also introduces a subtle fragility to the process.

[22] WhatsApp has also deployed key transparency-based verification on all iOS and Android devices [36]. However, they additionally support security code verification.

Short Authenticated Strings (SAS). Vaudenay [57], Laur et al. [33, 34], and Naor et al. [44], introduced SAS to authenticate high-bandwidth insecure channels via low-bandwidth authenticated out-of-band channels. This "security code" approach was later extended to two-party [46] and group key exchange [35]. The latter model was recently adapted to analyze the Olvid messenger [1]. Unlike [35,46], which use SAS and an authenticated out-of-band channel to validate a key exchange protocol transcript, we present security definitions for a group key exchange protocol based on in-band verification. Unlike in [1, 35], participants do not need to agree on unique identities for each other.

Rotem and Segev propose a formalism and construction for out-of-band group message authentication [50] in the context of secure group messaging. Naor, Rotem, and Segev [43] build upon this to provide out-of-band group key exchange.[23] Their construction requires a special group member (a leader) to have the ability to send authenticated messages to all other group members, which requires pre-existing agreement on unique participant identities and on who the leader is; these are difficult assumptions in a remote group setting. In contrast, we do not need such assumptions.

The ZRTP protocol [15] provides peer-to-peer E2EE using in-band SAS verification. However, justifying this through existing SAS security analyses is based on assuming the participants can authenticate each other through voice or video recognition and, thereby, form authenticated channels. This is an unreasonable assumption, as the participants may not know each other ahead of time. While [15, Section 15] suggests that video stream continuity through the SAS verification and the rest of the communication may suffice, this claim lacks formal analysis and is difficult to apply for groups, where some participants may not interact beyond the authentication phase (however, note that ZRTP does not support group key exchange for E2EE). In contrast, our protocol is provably secure without assuming authenticated channels, supports group communication, and is clear about what is needed of our human authentication functionality.

ZRTP also uses the secret key itself to derive the security code, rather than a list of public keys or key exchange transcript. This approach has a number of issues, however. First, it does not protect against insider attacks unless the key exchange is contributive. Second, as shown in [43, Section 4.2], this has the potential to undermine key indistinguishability if the code is not kept private.

On the (in)security of one-way ceremonies. Naor et al. describe how, while a one-way ceremony can be sufficient for out-of-band message authentication, it is not sufficient to build an out-of-band authenticated key exchange [43, Section 4.1]. To make this possible, they introduce an "out-of-band feedback signal" that allows the non-speaking participants to indicate to the speaking party (and all other members) that the ceremony has failed (having received a non-matching code, for example).

We remark that such a solution is not sufficient in the in-band setting without the use of pre-agreed upon context (e.g., who the speaker will be ahead of time). We demonstrate this with a generic attack against one-way ceremonies held in-band. Consider a one-way ceremony between a group of participants whereby, much like in the ceremony described by Naor et al., a special group member speaks the code to the remaining members, each of which is able to alert all members if they receive the wrong code (or one fails to appear). Regardless of the structure or contents of the security code, or even how it is tied to the key exchange, the adversary can perform the following attack. First, they schedule multiple underlying key exchanges between each member of the group and the rest of the group, appropriately proxying all messages. Next, the adversary triggers each party to act as the special group member (speaker), but suppresses the resulting messages sent by that party. Since each party considers themselves to be the speaker, they do not expect to receive a message from the others and, since no other party sends an error alert to them, they assume the ceremony has completed successfully.

Importantly, it is not possible to solve this problem through the key exchange, since an adversary can simply surface a different version of the key exchange to each party. It is solvable by having the members decide on who the leader is while on the call. To do so, however, requires two-way communication. A version of this ceremony that does not ask all members to speak the code, but simply asks for active confirmation from the non-speaking members (rather than only alerting on failures), does solve this particular issue. However, such confirmations are vulnerable to being replayed across sessions (as we demonstrate in Attack 4 of Section 1.4).

[23] A key advantage of their construction is its ability to provide "immediate key delivery," which ensures protocol success despite some participants dropping out. This comes at a cost trade-off between round complexity and the probability of detecting an active attack. Immediate key delivery is not a goal for us and we choose to focus our efforts elsewhere.