Integrating Causality in Messaging Channels

Shan Chen[1(✉)] and Marc Fischlin[2]

[1] Southern University of Science and Technology, Shenzhen, China
chens3@sustech.edu.cn
[2] Cryptoplexity, Technische Universität Darmstadt, Darmstadt, Germany
marc.fischlin@tu-darmstadt.de
https://www.cryptoplexity.de

Abstract. Causal reasoning plays an important role in the comprehension of communication, but it has been elusive so far how causality should be properly preserved by instant messaging services. To the best of our knowledge, causality preservation is not even treated as a desired security property by most (if not all) existing secure messaging protocols like Signal. This is probably due to the intuition that causality seems already preserved when all received messages are intact and displayed according to their sending order. Our starting point is to notice that this intuition is wrong.

Until now, for messaging channels (where conversations take place), both the proper causality model and the provably secure constructions have been left open. Our work fills this gap, with the goal to facilitate the formal understanding of causality preservation in messaging.

First, we focus on the common two-user secure messaging channels and model the desired causality preservation property. We take the popular Signal protocol as an example and analyze the causality security of its cryptographic core (the double-ratchet mechanism). We show its inadequacy with a simple causality attack, then fix it such that the resulting Signal channel is causality-preserving, even in a strong sense that guarantees post-compromise security. Our fix is actually *generic*: it can be applied to any bidirectional channel to gain strong causality security.

Then, we model causality security for the so-called message franking channels. Such a channel additionally enables end users to report individual abusive messages to a server (e.g., the service provider), where this server relays the end-to-end-encrypted communication between users. Causality security in this setting further allows the server to retrieve the necessary causal dependencies of each reported message, essentially extending isolated reported messages to message flows. This has great security merit for dispute resolution, because a benign message may be deemed abusive when isolated from the context. As an example, we apply our model to analyze Facebook's message franking scheme. We show that a malicious user can easily trick Facebook (i.e., the server) to accuse an

Shan Chen is affiliated with both the Research Institute of Trustworthy Autonomous Systems and the Department of Computer Science and Engineering of SUSTech.

M. Joye and G. Leander (Eds.): EUROCRYPT 2024, LNCS 14653, pp. 251–282, 2024.
https://doi.org/10.1007/978-3-031-58734-4_9

innocent user. Then we fix this issue by amending the underlying message franking channel to preserve the desired causality.

Keywords: Causality · Secure messaging · Signal · Message franking

1 Introduction

Causality deals with the relationship of cause and effect. In computer systems causality preservation should ensure that events are processed in the right order. This is a long-standing topic in the area of distributed computing, e.g., Lamport's seminal work on logical clocks [23] and follow-up works on determining consistent global snapshots [8] and state recovery [34]. The ideas in these works, e.g., the ability to reconstruct the global state from local information, are still valid today.

Causality preservation has meanwhile also entered the area of cryptography. In particular, it was recently identified as a desired security property for secure instant messaging protocols, as discussed *informally* in [32,36]. However, there the goal of causality preservation is quite *weak*: "implementations can avoid displaying a message before messages that causally precede it" [36]. This may seem correct at first glance as it borrows the same intuition from distributed computing for ordering events, but a closer look shows that such a guarantee is actually not sufficient for secure messaging (SM). The reason is that message dependencies are much more subtle than event dependencies: the user's comprehension of a received message may be influenced by *any* messages displayed before it, even if some of them are causally *independent*. We illustrate this with a classic example below.

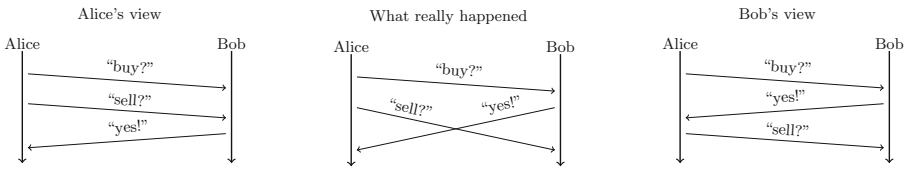

Fig. 1. Classic causality confusion example

As shown in Fig. 1, Alice asks Bob for investment advice using an instant messaging application. At first, Alice asks if she should buy a stock and Bob confirms, but Bob's response got delayed (e.g., due to network issues or attacks). From Alice's view, Bob remains silent, so Alice thinks he is currently offline. After a while, Alice tries to reach Bob again but this time she asks if she should sell the stock. Then, Alice receives Bob's response and mistakenly sells her stock.

It is worth noting that in the above example all messages are delivered and displayed in the correct order, so the causality confusion is *not* caused by out-of-order message display. The reason is that the message order cannot represent

the exact causal relations of the real communication. In particular, the "yes!" response from Bob does not tell Alice which of her messages he replied to. One may then be tempted to address this issue with a "reply-to" feature provided by some instant messaging applications, however, Bob did not know that he had to "reply-to" the "buy?" message because his view was not ambiguous at all (i.e., only the "buy?" message was received before his response). Even if users are required to "reply-to" all messages, which significantly hampers usability, this feature usually cannot handle a response that depends on *multiple* messages.

Therefore, to resolve or mitigate causality confusion, it is better (or at least as a useful complement) to enable SM applications to extract the necessary causal information from their "channel-layer" protocols (through which users transmit application messages). This idea is formulated as a causality-preserving property in our model, which roughly captures an SM channel user's ability to locally reconstruct the global causal relations of the communication. Note that such security is against active man-in-the-middle attacks, so it cannot be guaranteed by *unauthenticated* transport-layer protocols like TCP. Besides, our causality-preserving feature does not affect the *immediate decryption* property [1] usually required by SM channels. That is, when appended with the associated causal information, each received message can still be immediately decrypted upon receipt; meanwhile not only its sending order but also the exact message dependencies are reconstructed by the receiver.

Furthermore, compared to SM applications, it is probably more urgent and necessary to integrate causality in the so-called *message franking* schemes. Such a scheme additionally enables users to report abusive messages to the middle server who relays their end-to-end-encrypted communication. Clearly, the causal dependencies (i.e., the context) of an individually reported message is crucial for the server to determine if it is abusive.

For instance, a response to the question "what was the worst insult you have ever heard?" should be treated as benign, but it looks abusive when isolated from the context. A direct mitigation is to utilize timestamps that the server (e.g., Facebook) adds to each relayed message: the accused person can report the above question and argue that the seemingly abusive message is just a response to that question, as justified by their associated timestamps. However, this approach is not perfect, because timestamps reflect only the order of messages received by the server rather than the exact causal relations of the end-to-end conversation. For example, in Fig. 1 the server may still mistakenly view concurrent messages "sell?" "yes!" as sequential ones (i.e., as in either Alice's view or Bob's view). As another example, when Bob sends "my friend was insulted like this" followed by a message with abusive words, Alice can accuse Bob by reporting only the second message. Then, since in message franking only the message receiver (Alice) is allowed to report, the timestamp of the reported message does not help the server determine if Bob has ever sent a message right before the reported message.

In order to resolve causality issues in abuse reporting, one can enable the server to extract the entire (or necessary) context associated with the reported message. This is formulated as report causality preservation in our model.

1.1 Causality in Cryptographic Channels

Following previous work [1,20], we treat (two-party) SM channels as *bidirectional* channels. In this work, we focus on their causality-preserving property.

In the cryptographic literature, channels were often defined as a *unidirectional* primitive where one party only sends messages and the other party only receives. For this simplified setting, the desired channel security is usually modeled with respect to a cryptographic primitive called stateful authenticated encryption. This primitive was proposed by Bellare *et al.* [5] and later adopted or refined by follow-up works [6,21,22,27], mainly used to analyze the Transport Layer Security (TLS) record protocol. Recently, Marson and Poettering [26] initialized the formalization of bidirectional channels and their security, and showed how to securely combine two unidirectional channels to construct a bidirectional channel. Their results have later been extended to analyze multi-party broadcast channels [14], SM channels [20], and message-franking channels [19]. What all these approaches have in common is that they considered only channels on top of reliable networks (e.g., their constructions cease further functionality when a single message got lost). This however does not match the typical design of SM channels that could operate on *unreliable* networks, for which permanent message loss is possible. To tolerate message loss and meanwhile enable immediate decryption, Alwen *et al.* [1] extended the model for SM channels and applied it to analyze Signal's channel protocol, but they did not consider causality issues.

There were two formal analyses aiming to model causality for multi-party cryptographic channels [14,25], but neither is satisfactory even for two parties. In particular, [25] defines causality as implied by ciphertext integrity, which should not be the case for a well-defined causality notion, e.g., Signal is proved to achieve ciphertext integrity [1] but causality confusions can still occur (e.g., the example in Fig. 1). The other work [14] focuses on a different object called broadcast channel, but their security notion captures only the aforementioned *weak* causality preservation goal (i.e., to avoid displaying a message before messages that causally precede it). Besides, *neither* work handles message loss or immediate decryption. Therefore, both the proper model of causality preservation for SM channels and the provably secure constructions remain open.

The other setting we consider for causality preservation is secure abuse reporting (also known as message franking). Here secure messaging is extended to enable users to report abusive messages to a server (e.g., the service provider), who relays their encrypted communication. Message franking was named and first introduced by Facebook's end-to-end-encrypted message system [15]. Its rough idea is to add message commitments to the underlying SM channel and let the server tag the encrypted messages transmitted through it. Formal analysis of message franking was initiated by Grubbs *et al.* [18] and continued by follow-up works on attachment franking [12] and asymmetric message franking [35], all of which treat message franking as an unidirectional primitive. Recently, bidirectional message franking channels were modeled in [19]. However, prior works on message franking essentially treat reported messages *individually* so do not consider their causality.

1.2 Our Contributions

The main contribution of our work is a formal study of the *proper* causality preservation model for messaging channels. We focus on two settings: two-party secure messaging and message franking. In each setting, we define a security model for it and propose provably secure constructions by adding causality to a popular real-world protocol. We hope that our formal results can help to clarify the subtleties of causality issues and facilitate the integration of causality in messaging channels. More details are summarized as follows.

Modeling Causality Preservation for Bidirectional Channels. Intuitively, causality is preserved by a bidirectional channel if the communicating parties are able to locally reconstruct the global view of their conversation. Such a global view is formalized in Sect. 2 as a so-called *causality graph*, a bipartite graph where each vertex represents a sending or receiving action and each edge represents a message transmission. It can be viewed as a simplified two-party version of the multi-party communication graph defined in [25]. With such a causality graph, we model causality preservation for bidirectional channels in Sect. 4. To match the practical design of SM channels, our model incorporates two important aspects that were *not* considered by previous causality works:

- Our model is compatible with *unreliable* networks, i.e., tolerating message loss and out-of-order delivery;
- Our causality security in its strong version captures *post-compromise security*, i.e., causality can be recovered even after a state compromise if the adversary stays *passive* during recovery [10]; this property is critical for SM channels since here a session may last for a very long time (e.g., months).

Relations to Integrity Notions. So far, all previous works on causality preservation essentially defined it as implied by ciphertext integrity. However, as mentioned before, this should not be the case if causality preservation is properly defined. In Sect. 4.5, we show that our causality preservation notion is completely separate from ciphertext integrity, as expected. Note that causality preservation, however, implies plaintext integrity, as otherwise the attacker can manipulate the message dependencies by simply modifying the messages (and causality becomes meaningless if the associated messages can be changed).

Causality Preservation of TLS 1.3. Before applying our model to analyze Signal, we first investigate a simpler bidirectional channel — the TLS 1.3 record protocol [30]. Since mitigating causality confusion for TLS may not seem very important, we do not claim this as our main contribution and leave it in the full version [9]. Nevertheless, adding causality to the TLS channel turns out to be very simple and practical, making it appealing to identify suitable use cases (a toy example is described in the full version [9]).

Formally, we first show that the TLS 1.3 channel cannot preserve causality even in our basic model (i.e., with no post-compromise security and assuming reliable in-order message delivery). Our causality attack essentially reflects the causality confusion illustrated in Fig. 1. To address that, we propose efficient fixes that add necessary causal information to each transmitted message, such that the resulting *causal* TLS 1.3 channels provably achieve causality preservation. Thanks to reliable in-order message delivery, one only has to add the number of *consecutively* received ciphertexts, denoted by δ, along with each sent message. This elegant idea has already appeared in [25, Remark 5, p.79] for constructing causal channels in their model, but not yet applied to any real-world protocols. For TLS 1.3, we show that δ can be securely added as part of the message, of the associated data, or even of the local nonce; the former two are very practical.

Causality Preservation of Signal. In Sect. 5, we analyze Signal's channel protocol (the double-ratchet mechanism [28]) with our strong causality preservation model that captures unreliable network and post-compromise security. First, we show that the Signal channel also suffers from a similar causality attack as in the TLS case, which actually implies its insecurity even in our weak model. To fix it, we also add necessary causal information to each transmitted message. However, since Signal may operate on unreliable networks, transmitting only δ is not enough to derive all causal dependencies of the communication. We resolve this by using a first-in-first-out queue Q to record the entire causal information before each sent message. As transmitting all previous causal information may incur too much overhead (i.e., linear in the number of exchanged messages), we further show how Q can be shortened such that in common scenarios the overhead is small enough for practical use. The resulting causal Signal channel is proved to preserve strong post-compromise causality. It turns out that our proposed fix is *generic*, i.e., it can be applied to any bidirectional channel to provide strong causality security. Finally, we show a concrete way for SM applications to integrate causality in their application-layer user interfaces.

Modeling Causality Preservation for Message Franking Channels. In Sect. 6, we present our causality preservation model for message franking channels. It captures two types of attackers. The first type considers a malicious server (which relays the end-to-end-encrypted communication) against honest users. Our security notion for this type is called *channel causality preservation*, which captures the security of the underlying SM channel and is defined in the same way as for bidirectional channels described above. The second type considers a malicious user that tries to fool the reporting system by tampering with causality. Causality preservation against such attacks is modeled as *report causality preservation*, which guarantees that successfully received messages must be reportable and successfully reported messages must be honest and carry the correct causal information. Note that, unlike the first type, here the second-type attacker knows the secret state used to encrypt and decrypt messages.

Causality Preservation of Facebook's Message Franking. Finally, in Sect. 7 we apply our model to analyze Facebook's message franking scheme. First, we show that it does not preserve channel causality, as the same causality attack against Signal works here. Then, we show that the scheme does not preserve report causality either, even if it uses our causal Signal channel. This is because no causal information associated with the reported message is carried in the report. We fix this in our provably secure generic construction by adding and committing the missing causal information (kept in a queue similar to the Signal case). Our construction allows the defendant to prove with causality that the reported abusive message has been taken out of context.

1.3 Further Related Work

Alwen *et al.* [1] formalized the property of *immediate decryption*. This property says that the receiver of a message can decrypt a ciphertext obtained from the sender instantaneously upon arrival, even in settings with out-of-order delivery. Moreover, the recipient can also identify the ordinal number in the sequence of received messages. The notion has later been refined in [11,29]. Immediate decryption thus focuses on a functional property, with some weak aspects of reliable ordering of received messages at a party's site. The bilateral (or potentially multilateral) view of causality, capturing dependencies between sent and received messages in communication, is thus orthogonal.

Continuing the line of research about immediate decryption, Barooti *et al.* [2] defined the notion of *recovering with immediate decryption* (RID), as an extension of the notions in [7,13]. The receiver version of the RID notion, denoted as r-RID, demands that the receiver can detect if a previously received ciphertext has been maliciously injected by the adversary. The sender version, s-RID, requires that the sender can detect that the receiver has obtained such a malicious ciphertext. The noteworthy extension in [2] is that the authors consider communication channels with out-of-order delivery. While RID is primarily an integrity notion, the solutions in [2] themselves share the idea of including history information in the ciphertexts with our constructions—which ultimately can be traced back to [25]. Namely, in [2] the receiver transmits the list of received ciphertexts (for r-RID) or a hash thereof (for s-RID). Our security goal, however, and the details of our constructions are different: we do not consider active attack detection while they do not handle causality.

Formal security treatments of out-of-order delivery in cryptographic channels can be found in [6,22,31]. Recently, Fischlin *et al.* [16] defined a more fine-grained *robustness* property for channels over unreliable networks. Robustness complements the classical integrity notion and states that maliciously injected ciphertexts on the network cannot disturb the receiver's expected behavior. In contrast, causality addresses dependencies on the message level, thus aiming at a different scope. One could, nonetheless, integrate a robustness notion as in [16] on top, on the channel level. Indeed, the Signal protocol already has robustness built in, which follows as in [16] for QUIC, because the receiver's state remains unchanged for an illegitimate ciphertext.

2 Causality Graphs

In order to formally define the causality preservation security, we introduce the notion of a *causality graph* associated with an interactive communication (often called a session) between two parties, say Alice (A) and Bob (B). We follow the idea of multi-party communication graphs described in [25], but focus on the two-party case and extract the most relevant aspects from their notions.[1]

Intuitively, a causality graph unambiguously identifies all causal information, i.e., dependencies of sending and receiving actions, in the associated communication session. Note that here only *successful* receiving actions are considered in the graph, i.e., each receiving action corresponds to an accepted message. The graph is *not* static: it grows with ongoing communications within the session and always reflects all dependencies of already performed actions. Formally, we have the following definition for the two-party case.

Definition 1. *The causality graph $G = (V_A, V_B, E, <)$ associated with a two-party communication session is a bipartite graph with two strict (or irreflexive) total orders respectively on the disjoint vertex sets V_A, V_B, and a strict partial order on all vertices, where the notation $<$ is overloaded to denote all orders.*

Each vertex represents either a sending action (called a sending vertex) or a receiving action (called a receiving vertex) performed by some party and V_A, V_B respectively denote the vertex sets of party A, B. The edge set E consists of only directed edges from sending to receiving vertices, each edge representing the transmission of a message. The orders on V_A and on V_B are naturally defined according to the increasing occurrence times of the represented actions. The order on $V_A \cup V_B$ is the transitive closure of the orders on V_A, V_B and the order implied by the directed edges (i.e., $(x, y) \in E \Rightarrow x < y$).[2]

G is correct *if and only if 1) the above defined order on $V_A \cup V_B$ is a strict partial order and 2) each receiving vertex is connected to* exactly *one sending vertex and each sending vertex is connected to* at most *one receiving vertex.*

With the strict partial order on $V_A \cup V_B$, the above causality graph unambiguously identifies all dependencies of the already performed sending and receiving actions. We say two edges $(x_1, y_1), (x_2, y_2) \in E$ are *concurrent* if 1) they are in opposite directions (i.e., x_1, x_2 cannot both belong to V_A or to V_B) and 2) $y_1 \not< x_2$ and $y_2 \not< x_1$; the latter means x_1, y_1, x_2, y_2 cannot be totally ordered. Intuitively, two concurrent edges do not depend on each other. We also say a (sending) vertex is *isolated* if it is not connected to any edge, which could happen when the message has not been delivered or got lost during transmission.

A pictorial description of an example causality graph is given in Fig. 2 (left). In the dashed box, we see two pairs of concurrent edges: (a_1, b_3), (b_1, a_2) as well as (a_1, b_3), (b_2, a_3). An example of a non-concurrent edge pair is (a_5, b_5) from Alice to Bob together with (b_6, a_7) from Bob to Alice in the lower part, where

[1] We note that [25] defined a notion called *causal graph*. This looks similar but is actually for reliable networks, while our causality graph captures unreliable networks.

[2] This is actually the strict partial order derived from Lamport's logical clock [23].

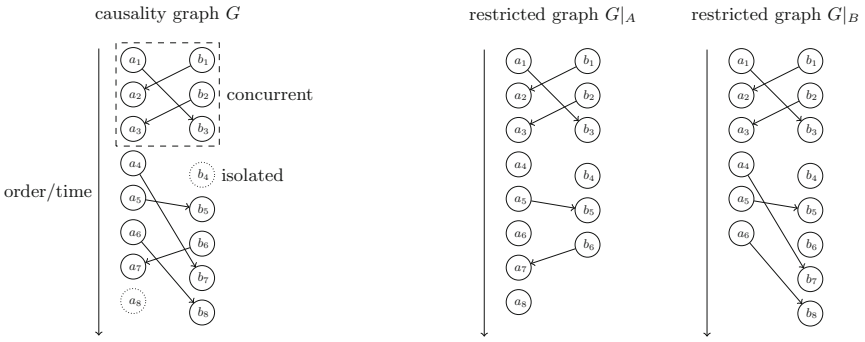

Fig. 2. An example causality graph G and the restricted graphs $G|_A, G|_B$ of Alice (left party) and Bob (right party).

the latter edge depends on the former one. The figure also shows two (dotted) isolated sending vertices a_8 and b_4.

Graph Addition. In order to model dynamic updates of the causality graph, we define a binary addition operation $+$ that inputs a graph and an action and outputs an updated graph. Let $(\mathtt{S}, P)$ denote a sending action of party $P \in \{A, B\}$. We write $G \leftarrow G + (\mathtt{S}, P)$ to express that G is updated by capturing $(\mathtt{S}, P)$, i.e., adding a new sending vertex v to the vertex set V_P (then v will be the *largest* vertex in V_P with respect to $<$). Let $(\mathtt{R}, P, i)$ denote a receiving action of party P, with the associated sending action represented by the i-th sending vertex $\bar{v}_i$ in $V_{\bar{P}}$, where $\bar{P} = \{A, B\} \setminus P$; here $\bar{v}_i$ exists because this sending action occurred before $(\mathtt{R}, P, i)$. Similarly, we write $G \leftarrow G + (\mathtt{R}, P, i)$ to express that G is updated by capturing $(\mathtt{R}, P, i)$: first add a new receiving vertex v to V_P and then add a directed edge $(\bar{v}_i, v)$.

Restricted Graph. Intuitively, the restricted graph $G|_P$ of party P captures the causality graph G restricted to P's view. Let v be the largest vertex in V_P. Formally, $G|_P$ is a subgraph of G that consists of v, all vertices in $V_A \cup V_B$ that are smaller than v, and all edges between those vertices; this is also known as the *v-prefix* of G as defined in [26]. $G|_P$ can be efficiently derived from G.

Note that $G|_P$ excludes any edge (and its receiving vertex) that is concurrent to, or larger than, the last edge from $\bar{P}$ to P. Consider the example causality graph G shown in Fig. 2. The restricted graph $G|_A$ of Alice excludes edges $(a_4, b_7), (a_6, b_8)$ (and vertices b_7, b_8) because they are concurrent to (b_6, a_7) (which is the last edge from Bob to Alice). This reflects the fact that Alice does not know whether the messages sent at a_4, a_6 have been delivered to Bob because she has not received any response regarding those messages yet. Alice at a_7 received a message sent from Bob at b_6; however, this receiving action only confirms the delivery of Alice's messages sent at a_1, a_5 but not those sent at a_4, a_6, since the latter are received after b_6. Similarly, the restricted graph $G|_B$

of Bob excludes edge (b_6, a_7) and vertex a_7. It also does not include vertex a_8 because it is not smaller than b_8 (the largest vertex in V_B); this reflects the fact that Bob is not yet aware of Alice sending at a_8.

3 Preliminaries

Notations. Let $\perp$ denote an invalid element. The output of a function or algorithm is all $\perp$(s) if any of its input is $\perp$. Let . denote the member access operation, e.g., $a.x$ denotes the x element of a. However, in the figures that depict the security experiments and protocols shown later, the state prefixes are omitted for simplicity, e.g., if a state st contains an element x then we simply write x instead of $st.x$.

In the full version [9], we recall the definitions of authenticated encryption with associated data (AEAD), message authentication codes (MACs), and commitment schemes with verification, as well as their corresponding advantage measures that this work focuses on: $\mathbf{Adv}^{\mathsf{auth}}_{\mathsf{AEAD}}$, $\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{MAC}}$, and $\mathbf{Adv}^{\mathsf{v\text{-}bind}}_{\mathsf{CS}}$.

4 Bidirectional Channels and Causality Preservation

4.1 Bidirectional Channels

A bidirectional channel allows two parties (or users), Alice (A) and Bob (B), to securely communicate with each other, where each party $P \in \{A, B\}$ can send messages to the other party $\bar{P} = \{A, B\} \setminus P$, and receive messages sent by $\bar{P}$. For security reasons, the sending party transforms messages to ciphertexts before transmitting them and the ciphertexts are later transformed back to messages by the receiving party. Both parties can keep states across their sending and receiving actions. Formally, we have the following definition based on the bidirectional channel notion proposed by [26].

Definition 2. *A bidirectional (cryptographic) channel is a three-tuple* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Snd}, \mathsf{Rcv})$ *associated with a key space* $\mathcal{K}_{\mathsf{Ch}}$*, a state space* $\mathcal{ST}$*, a message space* $\mathcal{M}$*, and an index space* $\mathcal{I}$*:*

$\mathsf{Init}(P, k) \to st_P$*: takes* $P \in \{A, B\}$*,* $k \in \mathcal{K}_{\mathsf{Ch}}$*, and outputs the initial state of* P*;*

$\mathsf{Snd}(P, st, m) \xrightarrow{\$} (st', c)$*: takes* $P \in \{A, B\}$*,* $st \in \mathcal{ST}$*,* $m \in \mathcal{M}$*, and outputs an updated state* $st' \in \mathcal{ST}$ *and a ciphertext* $c \in \{0,1\}^*$*;*

$\mathsf{Rcv}(P, st, c) \to (st', m, i)$*: takes* $P \in \{A, B\}$*,* $st \in \mathcal{ST}$*,* $c \in \{0,1\}^*$*, and outputs an updated state* $st' \in \mathcal{ST}$ *and a message* $m \in \mathcal{M} \cup \{\perp\}$ *with index* $i \in \mathcal{I}$*.*

Correctness requires that each party outputs the messages sent by the other party together with the correct index that indicates their sending order.

We say a party *accepts* a message m (and the ciphertext c) if Rcv processing c is successful, i.e., it outputs $m \neq \perp$. If the channel runs over an unreliable network, we follow [1] to require that (i) state st remains *unchanged* if Rcv

outputs $m = \perp$; (ii) Rcv never accepts two messages with the *same* index; and (iii) index i can be efficiently extracted from the ciphertext c (denoted by $c.i$).

Note that the message index i can be either a simple ordinal number in $\mathbb{N}$ that matches a send counter, or of any form as long as the indices are strictly ordered. For instance, in the SM syntax of [1], an index is a two-tuple that consists of an epoch number and a send counter within that epoch. However, due to the bijective mapping between indices and ordinals, our definitions for simplicity do not differentiate them explicitly.

Definitional Differences From [26]**.** First, our channel algorithms have the acting party's identity as an explicit input to capture the *different* behaviors of the communicating parties when running the same algorithm with the same inputs, e.g., TLS client and server use different components of the same session key (part of the input state) for encryption (in Snd) and decryption (in Rcv). Furthermore, for conciseness our Snd and Rcv algorithms do not take as input unencrypted application-level associated data, i.e., channel parties require the entire input message to be encrypted, which is often the case for real-world bidirectional channels (e.g., TLS 1.3, Signal, etc.). As we will show, there may be some associated data formed by the bidirectional channels and authenticated by their underlying authenticated encryption schemes, but such associated data is not specified by the channel users. However, it is easy to extend our definition to capture the application-level associated data if desired. Finally, our Rcv algorithm additionally outputs an index i to determine the sending order of received messages, which is necessary to model out-of-order delivery or message loss, but often omitted if the channel is over a reliable in-order network.

4.2 Local Graph and Its Update Function

Our security definitions utilize the notion of a *local graph* G_P to represent the causal information derived by a party P. The local graph can be constructed from the party's local protocol execution. Causality preservation of a channel should imply that each party's local graph always matches its restricted graph, i.e., $G_P = G|_P$. Intuitively, this means that local protocol execution is consistent with the party's expected view on causality: What the parties knows about the causality structure is accurate (up to what can be guaranteed).

A local graph update function localG is a function invoked after each successful Rcv execution. Function localG inputs a local graph and the Rcv execution's transcript T_{Rcv} and outputs an updated local graph. Note that the transcript consists of all the input, output, random coins, internal states, etc., used in the considered Rcv execution. The intuition behind localG is to update the local graph with the causal information extracted from the successful receiving action. Such a function is necessary because extracting causal information from received ciphertexts is the only way for a party P to correctly order the other party $\bar{P}$'s sending and receiving actions in its local graph G_P, as P does not have access to $\bar{P}$'s view. Furthermore, we define localG to concern only receiving actions because successful sending actions can be trivially added to the local graph in an unambiguous way, which is denoted by $G_P \leftarrow G_P + \mathsf{s}$.

4.3 Causality Preservation

Now, we formally define the security notion of *causality preservation (CP)*. The idea is that the adversary wins if it makes some party's local graph G_P deviate from the restricted graph $G|_P$, i.e., if the party's internal view on causality differs from the actual (local) view. We note that the adversary also wins (event Bad below) if it makes the receiver accept a malicious message, either one that has not been sent (if Ch is designed for unreliable networks) or one that has not been sent or is delivered in wrong order (if Ch is designed for reliable in-order networks). The former event occurs if the receiver outputs a message m with index i which has not been put on the wire, and the latter event further checks if the index i is as expected. Note that in the first case we cannot stipulate more since transmissions may get lost or be delivered later. Augmenting the security game by the Bad events ensures that the content of the message remains intact, thus guaranteeing that responses correspond to the right information.

Security Experiment. In Fig. 3, we depict the security experiment (or game) for causality preservation $\mathbf{Exp}^{\mathsf{cp}}_{\mathsf{Ch},\mathsf{localG},\mathcal{A}}(1^\lambda)$ that is executed between a challenger and an adversary $\mathcal{A}$. The experiment is associated with a bidirectional channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Snd}, \mathsf{Rcv})$ and a local graph update function localG.

$\mathbf{Exp}^{\mathsf{cp}}_{\mathsf{Ch},\mathsf{localG},\mathcal{A}}(1^\lambda)$:
1: $k \xleftarrow{\$} \mathcal{K}_{\mathsf{Ch}}$
2: $st_A \leftarrow \mathsf{Init}(A, k)$
3: $st_B \leftarrow \mathsf{Init}(B, k)$
4: $s_A, s_B, r_A, r_B \leftarrow 0$
5: $G, G_A, G_B \leftarrow \varepsilon$
6: $\mathcal{R} \leftarrow \varnothing$
7: $\mathcal{A}^{\mathsf{Send},\mathsf{Recv}}$
8: terminate with 0

$\mathtt{Send}(P, m)$:
1: $(st_P, c) \xleftarrow{\$} \mathsf{Snd}(P, st_P, m)$
2: if $c = \bot$ then return $\bot$
3: $G \leftarrow G + (\mathtt{S}, P)$, $G_P \leftarrow G_P + \mathtt{S}$
4: add (P, s_P, m, c) to $\mathcal{R}$, $s_P \leftarrow s_P + 1$
5: return c

unreliable networks:
$\mathsf{Bad} = [(\bar{P}, i, m, \cdot) \notin \mathcal{R}]$
reliable in-order networks:
$\mathsf{Bad} = [(\bar{P}, i, m, \cdot) \notin \mathcal{R}$ or $i \neq r_P]$

$\mathtt{Recv}(P, c)$:
1: $(st_P, m, i) \leftarrow \mathsf{Rcv}(P, st_P, c)$ // T_{Rcv}: transcript
2: if $m = \bot$ then return $\bot, \bot$
3: if Bad then
4: | terminate with 1 ($\mathcal{A}$ wins)
5: $G \leftarrow G + (\mathtt{R}, P, i)$
6: $G_P \leftarrow \mathsf{localG}(G_P, T_{\mathsf{Rcv}})$
7: if $G_P \neq G|_P$ then
8: | terminate with 1 ($\mathcal{A}$ wins)
9: delete $(\bar{P}, i, \cdot, \cdot)$ from $\mathcal{R}$, $r_P \leftarrow r_P + 1$
10: return m, i

Fig. 3. Security experiment for causality preservation

In the beginning, the challenger samples a random channel key k and calls Init with it to derive the initial states. All the states used in the game are also properly initialized, where in particular s_A, s_B, r_A, r_B are used to count sending and receiving actions. Then, $\mathcal{A}$ is given access to two oracles $\mathtt{Send}$ and $\mathtt{Recv}$:

$\mathtt{Send}$ takes a party identity and a message, calls Snd on the input message, updates the graphs, records the message, and outputs the derived ciphertext. Note that for reliable in-order networks when a receiving action fails the state st_P may be set to $\bot$ by Rcv, and if so $\mathsf{Snd}(P, st_P, \cdot)$ will always output $(\bot, \bot)$.

$\mathtt{Recv}$ takes a party identity and a ciphertext and calls Rcv on the input ciphertext. If the accepted message triggers the Bad event discussed above, $\mathcal{A}$ wins. Otherwise, the party's local graph G_P and the (global) causality graph G

are updated. Then, $\mathcal{A}$ wins if the local graph does not match the restricted graph. Finally, the oracle removes the accepted message from the record and outputs the message with its index.

Advantage Measure. The advantage is defined as $\mathbf{Adv}^{\mathsf{cp}}_{\mathsf{Ch},\mathsf{localG}}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\mathsf{cp}}_{\mathsf{Ch},\mathsf{localG},\mathcal{A}}(1^\lambda) \Rightarrow 1]$ for any arbitrary localG. We say a bidirectional channel Ch preserves causality (or is CP-secure) if one can *construct* an efficiently computable function $\mathsf{localG}^\star$ such that, for any efficient adversary $\mathcal{A}$, the advantage $\mathbf{Adv}^{\mathsf{cp}}_{\mathsf{Ch},\mathsf{localG}^\star}(\mathcal{A})$ is negligible.

The above security definition may look a bit elusive due to its reliance on the *constructibility* of $\mathsf{localG}^\star$ (which may not be unique), but the intuition is not complicated. Note that constructibility is a stronger requirement than existence because an existing function may be very hard to find (e.g., a function to output hash collisions). By definition, each party in a CP-secure channel can use $\mathsf{localG}^\star$ to extract all correct causal information associated with an ongoing session in the presence of an active attacker, which is impossible for an insecure channel due to the non-constructibility (or even non-existence) of $\mathsf{localG}^\star$.

Note that a CP-secure channel only guarantees that each party is *in principle* able to derive *all* causal information captured by its restricted graph, which corresponds to the constructibility of some $\mathsf{localG}^\star$. However, this does not imply that all correct causal information is indeed derived and utilized by the channel parties, e.g., they may use arbitrary functions to extract the necessary portion of causal information. This actually gives the practical channel constructions more flexibility for utilizing causality, i.e., it may be sufficient for a party to extract only *partial* causal information (rather than the entire local graph) to perform its causality-related functionality (see the TLS analysis in the full version [9] for example). In the future sections, we will illustrate in our analysis how exactly causality can be utilized to improve security for our proposed constructions.

4.4 Causality Preservation with Post-compromise Security

The above basic causality preservation notion is sufficient to analyze secure connection protocols like TLS 1.3 (see the full version [9]), for which state corruption leads to no security.[3] However, post-compromise security is an important concern for secure messaging (SM) protocols like Signal, since their sessions typically last for a long time (e.g., months). In order to capture this type of bidirectional channels, we define the notion of *strong causality preservation (SCP)* that recovers security after state compromise (and defaults to the basic weaker notion for uncompromised executions). Here for simplicity only unreliable networks are considered, as popular practical SM protocols like Signal usually do not assume reliable in-order message delivery.

[3] For secure connection protocols, our work focuses on their security within a basic connection, where no post-compromise security is guaranteed, but such protocols (e.g., TLS 1.3) could achieve post-compromise security across resumed sessions [33].

Epochs. In order to formalize post-compromise security, we follow the prior work to associate each party with a sequence of incrementing epochs $t = 0, 1, 2, \ldots$ that represents consecutive time periods. Each transmitted message and ciphertext are also associated with the same epoch as that of the party when it sent them. We assume that the epoch number t is part of the party's state st_P (denoted by $st_P.t$) and can be efficiently extracted from the ciphertext c (denoted by $c.t$). Then, for any ciphertext c *accepted* by a party P, we assume that $c.t \leq st_P.t + 1$. We will see that Signal satisfies the above assumptions. Finally, we let $(G_P)_{\geq t}$ and $(G|_P)_{\geq t}$ respectively denote subgraphs of G_P and $G|_P$ that consist of only vertices (and edges between them) created at epochs larger than or equal to t.

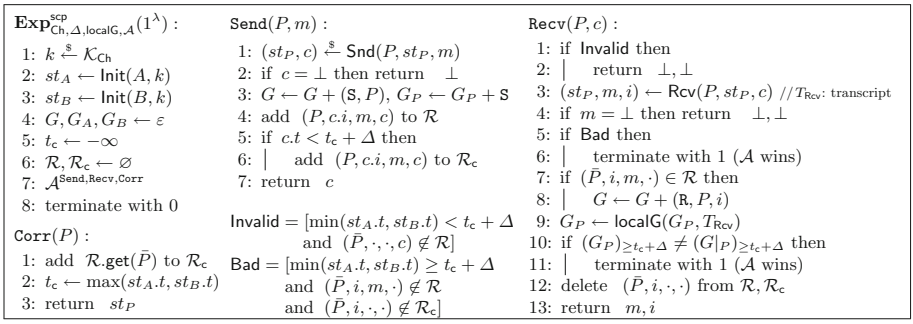

Fig. 4. Security experiment for strong causality preservation

Security Experiment. In Fig. 4 we depict the security experiment (or game) for strong causality preservation $\mathbf{Exp}^{\mathsf{scp}}_{\mathsf{Ch},\Delta,\mathsf{localG},\mathcal{A}}(1^\lambda)$ that is executed between a challenger and an adversary $\mathcal{A}$. The experiment is additionally associated with a parameter $\Delta \geq 0$ that indicates how fast (in terms of epochs) parties recover from state compromise. Intuitively, strong causality preservation guarantees that even if at some epoch a party is corrupted, after Δ epochs the channel protocol resurrects causality again.

The experiment is more complicated than the CP experiment due to state compromise. In the beginning, the challenger initializes two additional states, t_{c} that stores the most recent (i.e., largest) compromised epoch and $\mathcal{R}_{\mathsf{c}}$ that records the compromised messages (with the corresponding ciphertexts). Then, $\mathcal{A}$ is given oracle access to `Send`, `Recv`, `Corr`, where `Corr` is for state corruption.

`Corr` takes a party identity and outputs the party's current state; it also records all the outstanding messages sent by the other party as compromised (i.e., adding them to $\mathcal{R}_{\mathsf{c}}$) and updates t_{c}.

`Send` works as before except that: if the party is still recovering from state compromise, i.e., $c.t < t_{\mathsf{c}} + \Delta$, then the sent message and ciphertext are recorded as compromised.

`Recv` becomes more complicated to handle corruption, but it downgrades to the `Recv` oracle in the CP experiment when no corruption occurs (then $t_c = -\infty$ and $\mathcal{R}_c = \varnothing$). In the beginning, the $\mathsf{Invalid}$ condition is checked, which ensures that the adversary performs *passively* during channel recovery (i.e., no malicious ciphertext can be processed when either party's current epoch is less than $t_c + \Delta$). Then, if the ciphertext is successfully transformed to a message (i.e., the message is accepted), the Bad event is checked. Bad occurs if after recovery a party accepts a malicious message that was neither sent by the other party nor associated with a compromised epoch, and hence in this case $\mathcal{A}$ wins. Otherwise, the local graph G_P and (global) causality graph G are updated, where the latter is updated only when the accepted message is not modified since message dependencies captured by G are meaningless without the correct messages. Then, $\mathcal{A}$ wins if the after-recovery subgraph of either party's local graph $(G_P)_{\geq t_c+\Delta}$ does not match that of the party's restricted graph $(G|_P)_{\geq t_c+\Delta}$. Finally, the oracle removes the accepted message from the records and outputs the message with its index.

We remark that our model does *not* capture forward secrecy for causality. The key observation is that, even after state recovery, the part of a causality graph that corresponds to a previous uncompromised epoch may still be affected by a compromised message that carries malicious causal information. However, causality for already received messages is still guaranteed upon corruption.

Advantage Measure. The advantage is defined as $\mathbf{Adv}^{\mathsf{scp}}_{\mathsf{Ch},\Delta,\mathsf{localG}}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\mathsf{scp}}_{\mathsf{Ch},\mathsf{localG},\Delta,\mathcal{A}}(1^\lambda) \Rightarrow 1]$ for any arbitrary localG. We say a bidirectional channel Ch preserves Δ-strong causality (or is Δ-SCP-secure) if one can *construct* an efficiently computable function $\mathsf{localG}^\star$ such that, for any efficient adversary $\mathcal{A}$, the advantage $\mathbf{Adv}^{\mathsf{scp}}_{\mathsf{Ch},\Delta,\mathsf{localG}^\star}(\mathcal{A})$ is negligible. Similarly, a Δ-SCP-secure channel also guarantees that each party is *in principle* able to derive *all* causal information captured by its restricted graph in epochs after recovery, but parties may choose to extract only *partial* causal information.

SCP $\Rightarrow$ CP and CP $\not\Rightarrow$ SCP. For SCP $\Rightarrow$ CP, we note that SCP downgrades to CP if the adversary makes no corruption query, in which case $t_c = -\infty$ and $\mathcal{R}_c = \varnothing$. The other direction is not true, e.g., causal TLS 1.3 channels (details in the full version [9]) offer no post-compromise security.

4.5 Relations to Integrity Notions

Our (S)CP notions are clearly orthogonal to confidentiality (i.e., causal relations can be simply observed by a network attacker), but one may think of them as complements to integrity. We show that this is not quite the case.

First, in Fig. 5 we formalize the security experiments of plaintext integrity (INT-PTXT) and ciphertext integrity (INT-CTXT) for bidirectional channels.[4]

[4] [26] initialized the formal security definitions for bidirectional channels, but their notions do not capture unreliable networks.

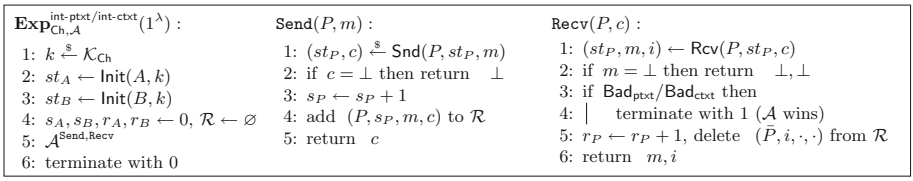

Fig. 5. Security experiments for plaintext and ciphertext integrity, where $\mathsf{Bad}_{\mathsf{ptxt}} = \mathsf{Bad}$ as defined in Fig. 3, $\mathsf{Bad}_{\mathsf{ctxt}} = [(\bar{P}, i, \cdot, c) \notin \mathcal{R}]$ for unreliable networks and $\mathsf{Bad}_{\mathsf{ctxt}} = [(\bar{P}, i, \cdot, c) \notin \mathcal{R}$ **or** $i \neq r_P]$ for reliable in-order networks.

Their advantage measures are defined naturally and denoted by $\mathbf{Adv}^{\mathsf{int\text{-}ptxt}}_{\mathsf{Ch}}(\mathcal{A})$ and $\mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathsf{Ch}}(\mathcal{A})$ respectively.

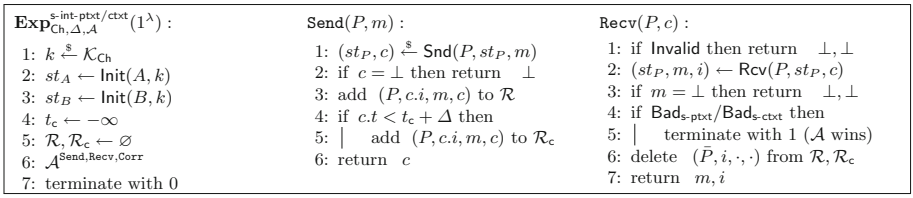

Fig. 6. Security experiments for strong plaintext integrity and strong ciphertext integrity, where $\mathtt{Corr}$, $\mathsf{Invalid}$ and $\mathsf{Bad}_{\mathsf{s\text{-}ptxt}} = \mathsf{Bad}$ are defined in Fig. 4 and $\mathsf{Bad}_{\mathsf{s\text{-}ctxt}} = [\min(st_A.t, st_B.t) \geq t_{\mathsf{c}} + \Delta$ **and** $(\bar{P}, i, \cdot, c) \notin \mathcal{R}$ **and** $(\bar{P}, i, \cdot, \cdot) \notin \mathcal{R}_{\mathsf{c}}]$.

Then, in Fig. 6 we define the security experiments for strong plaintext integrity (S-INT-PTXT) and strong ciphertext integrity (S-INT-CTXT) that offer post-compromise security for bidirectional channels. Similarly, we denote their advantage measures by $\mathbf{Adv}^{\mathsf{s\text{-}int\text{-}ptxt}}_{\mathsf{Ch},\Delta}(\mathcal{A})$ and $\mathbf{Adv}^{\mathsf{s\text{-}int\text{-}ctxt}}_{\mathsf{Ch},\Delta}(\mathcal{A})$ respectively.

To clarify the relationship of the above two notions, we define a notion called *robust correctness* (ROB-CORR) to capture correctness in a robust sense: after state recovery, decrypting ciphertexts created in a compromised epoch and decryption failure do not affect the correctness requirement, i.e., an honest ciphertext is always decrypted to the original message and index.[5] Its security experiment is the same as Fig. 6, except that the Bad event is replaced by $\mathsf{Bad}_{\mathsf{rob\text{-}corr}} = [\min(st_A.t, st_B.t) \geq t_{\mathsf{c}} + \Delta$ **and** $(\bar{P}, i, \cdot, c) \in \mathcal{R}$ **and** $(\bar{P}, i, m, \cdot) \notin \mathcal{R}$ **and** $(\bar{P}, i, \cdot, \cdot) \notin \mathcal{R}_{\mathsf{c}}]$. The advantage measure is denoted by $\mathbf{Adv}^{\mathsf{rob\text{-}corr}}_{\mathsf{Ch},\Delta}(\mathcal{A})$.

In the full version [9], we investigate the relations among the above integrity notions and our (S)CP notions; the results are summarized in Fig. 7.

[5] This notion is loosely connected to the idea behind the robust notion for unreliable channels recently put forward in [16], namely that malicious ciphertexts do not disturb the expected behavior. However, in our case the notion is closer to a correctness property after recovery. A similar correctness security notion was also defined in [1].

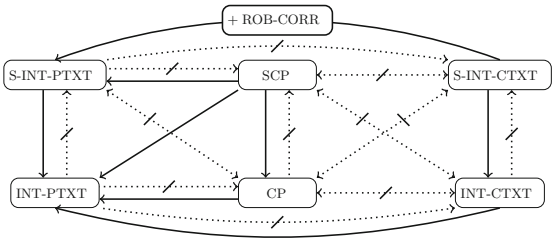

Fig. 7. Notion relations. Solid arrows mean an implication, dotted (crossed out) arrows mean a separation.

5 Causality Preservation of Signal

In this section, we analyze causality preservation of the Signal protocol [24,28]. We focus on its double-ratchet component [28] without considering the X3DH key agreement [24] used to derive the initial shared key. First, we show that Signal as a bidirectional channel does not even achieve the basic CP security. Then, we propose simple fixes to construct SCP-secure causal Signal channels and describe a potential user interface for the SM applications to display the causal dependencies to end users.

5.1 The Signal Channel and Its Insecurity

The Signal Channel. According to our defined syntax (see Definition 2), we can view Signal as a bidirectional channel, denoted by $\mathsf{Ch}_{\mathrm{Signal}}$. Here we briefly summarize its main cryptographic design, and refer to the full version [9] for a more detailed description of the Signal channel based on its core building blocks.

Signal performs a so-called *continuous key agreement (CKA)* protocol to generate a series of shared secrets, such that after state compromise the channel parties are able to recover security with a *fresh* shared secret. Parties in the Signal channel send and receive messages in *alternate* epochs, with odd epochs for Alice to send and Bob to receive, and even epochs for Bob to send and Alice to receive. Therefore, concurrent messages sent by different parties are associated with *distinct* epochs. Recall that in Sect. 4.4 we assume each party P keeps the epoch number t in its local state st_P and the associated epoch number can be efficiently extracted from the ciphertext; this is the case for Signal.[6]

The epoch numbers of both parties are initialized as 0. For each party P, its epoch number $st_P.t$ is incremented from t to $t+1$ in two cases: (1) after P receives from the other party a message with epoch number $t+1$ (e.g., when $st_B.t = 0$ and Bob receives a message associated with epoch $t = 1$, Bob updates $st_B.t = 1$); or (2) before P sends a message while t is not the epoch for P to send

[6] Actually, Signal exploits the uniqueness of the latest CKA message (authenticated but not encrypted, as shown in the full version [9]) to index epochs. For simplicity, we follow [1] to assume an explicit epoch number is used.

(e.g., when $st_A.t = 2$ and Alice wants to send a message, the epoch number is incremented to $st_A.t = 3$ because Alice can only send messages in odd epochs). This design matches our assumption in Sect. 4.4 that each bidirectional channel party P accepts only ciphertexts with epoch number $\leq st_P.t + 1$.

The above CKA also provides *forward secrecy*, which for Signal roughly means that state corruption does not affect the security of the (encrypted) messages already transmitted in previous epochs. Actually, forward secrecy guaranteed by Signal is more fine-grained, i.e., even within the *same* epoch the already sent messages remain safe. To achieve such security, each party in Signal further updates its sending (or receiving) key after each sending (or receiving) action, such that past keys cannot be derived from new keys.

The message index of $\mathsf{Ch}_{\text{Signal}}$ is hence a two-tuple (t, s), where t is the epoch number and s is the sent message counter within epoch t.

Causality Insecurity of $\mathsf{Ch}_{\text{Signal}}$. We can follow the idea reflected in Fig. 1 to construct an efficient adversary $\mathcal{A}$ against causality preservation of $\mathsf{Ch}_{\text{Signal}}$. First, $\mathcal{A}$ samples a random bit $b \xleftarrow{\$} \{0,1\}$. Then, consider the following queries for any three messages $m_1, m_2, m_3 \in \mathcal{M}$: ① $c_1 \xleftarrow{\$} \mathtt{Send}(A, m_1)$, ② $(m_1, (1,0)) \leftarrow \mathtt{Recv}(B, c_1)$, ③ $c_2 \xleftarrow{\$} \mathtt{Send}(A, m_2)$, ④ $(m_2, (1,1)) \leftarrow \mathtt{Recv}(B, c_2)$, ⑤ $c_3 \xleftarrow{\$} \mathtt{Send}(B, m_3)$, ⑥ $(m_3, (2,0)) \leftarrow \mathtt{Recv}(A, c_3)$. If $b = 0$, $\mathcal{A}$ runs ①②③④⑤⑥; otherwise $b = 1$, $\mathcal{A}$ runs in a different order: ①②③⑤④⑥. These two cases are depicted in Fig. 8.

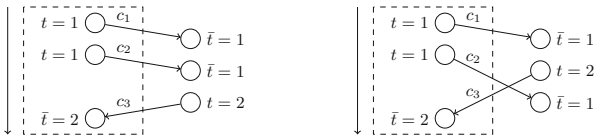

Fig. 8. Causality attack against Signal. Each ciphertext contains the epoch t for sending actions and the obtained epoch value $\bar{t}$ for receiving actions. The send counters are irrelevant for the attack and are omitted. The adversary chooses one of the execution flows randomly. Then, Alice's views (in the dashed boxes) in both cases are identical, whereas Alice's restricted graphs are different: the right hand side does not contain Bob's last vertex.

Clearly, the above two cases result in two different causality graphs (and different restricted graphs for Alice): in the left world ($b = 0$) Bob sent m_3 after receiving m_2 but in the right world ($b = 1$) that is not the case. Note that in both worlds Bob has received m_1 before sending m_3, so m_3 must belong to epoch $t = 2$.[7] Since c_3 carries no information about whether m_2 has been received, both worlds look identical to Alice. (This can be verified by checking the detailed

[7] Note that if Bob sends a message m before receiving any messages from Alice, then this message m belongs to epoch $t = 0$.

description of $\mathsf{Ch}_{\mathrm{Signal}}$ in the full version [9].) Therefore, $G_A \neq G|_A$ happens with probability at least 1/2, i.e., $\mathbf{Adv}^{\mathsf{cp}}_{\mathsf{Ch}_{\mathrm{Signal}},\mathsf{localG}}(\mathcal{A}) \geq 1/2$ for any possible update function localG. By definition, $\mathsf{Ch}_{\mathrm{Signal}}$ does not preserve causality.

5.2 Integrating Causality in Signal

Since Signal allows for out-of-order message delivery and message loss, transmitting only the δ value (i.e., the number of consecutively accepted messages before the sent message, more details discussed in the full version [9]) as for TLS is not enough to reconstruct the full causal relations. In order for the parties to build the correct restricted graph, along with each sent message the entire causal information before this message (that has not been known by the receiving party) has to be transmitted. We store this information in a queue Q (with the usual methods enq, deq, and front to enqueue and dequeue elements, and to read the front element without dequeuing it). Then, we propose a so-called *message-borne* causal Signal channel, indicating where Q is borne. Analogously, one can also construct an *associated-data-borne* causal Signal channel, by authenticating Q as part of the associated data rather than encrypting it.[8]

A Generic Causal Channel Compiler. In Fig. 9, we show a *generic* compiler that transforms an arbitrary bidirectional channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Snd}, \mathsf{Rcv})$ into a message-borne causal channel Ch^m. In particular, when Ch is instantiated with $\mathsf{Ch}_{\mathrm{Signal}}$, we get the message-borne causal Signal channel $\mathsf{Ch}^m_{\mathrm{cSignal}}$.

As shown in Fig. 9, Ch^m keeps indices i_S, i_R and queue Q as three additional states and encrypts the latter two states with the sent message. Formally, Q is a (first-in-first-out) queue that records a sequence of actions before the sent message in their correct time order: each action is recorded as the *index* of the associated sent or received message. We require that one can distinguish a sending index from a receiving index. Clearly, the receiving party is able to construct the correct restricted graph if *all* actions before the sent message are recorded in Q. However, this may incur too much overhead, e.g., a Signal communication session may last for months and hence involve many actions.

To mitigate overhead, we use indices i_S, i_R to update Q such that it records only the actions performed by party P but whose delivery has not yet been confirmed, i.e., P has not accepted any ciphertext sent from $\bar{P}$ that confirms the delivery of those actions. Let i_S denote, in P's view, the largest index of messages accepted by $\bar{P}$, then Q only needs to record P's actions after its i_S-th sending action, because earlier actions have been recorded and transmitted along with the sent messages accepted by $\bar{P}$. For instance, consider the message sent by Bob at b_6 in Fig. 2. This message has index 4 and queue Q consists of the (sending) message indices associated with b_3, b_4, b_5, i.e., $Q = (\bar{1}, 3, \bar{3})$ (where $\bar{i}$ indicates a receiving index), because the received message at b_5 already confirmed the

[8] As far as we know, the associated data is rarely used by instant messaging services for handling application-level data, so the message-borne version seems easier to understand and implement. It also matches our bidirectional channel syntax well.

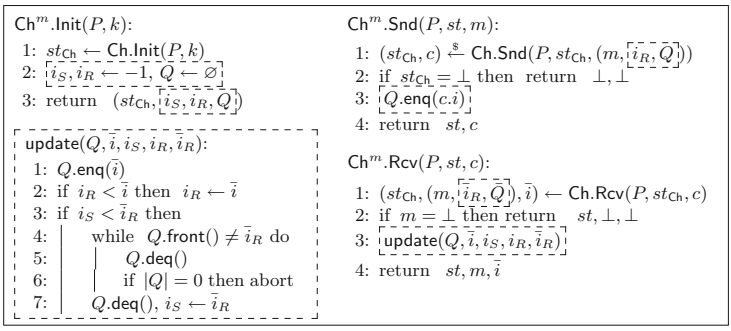

Fig. 9. The message-borne causal channel Ch^m (with dashed boxes highlighting the added causality-related operations). It deploys a queue Q and two indices i_S, i_R whose current values are always kept in the augmented state $st = (st_{\mathsf{Ch}}, i_S, i_R, Q)$. Barred values represent the data output by the receiver of the underlying channel (as opposed to internal states). The value $\bar{Q}$ is not returned by Rcv, but it is part of the Rcv transcript T_{Rcv} so can be used by localG to update the local graph. When $\mathsf{Ch} = \mathsf{Ch}_{\mathrm{Signal}}$, message indices are of the form (t, s) and ordered lexicographically (with -1 denoting a minimum).

delivery of messages sent at b_1 and b_2. In order to easily update i_S, we transmit an additional state i_R of P that records the largest index of accepted messages sent by $\bar{P}$, then i_S can be updated by comparing to $\bar{i}_R$ (i.e., the largest index of $\bar{P}$'s accepted messages sent by P) decrypted from ciphertexts sent by $\bar{P}$. This generalizes the idea of δ value, where it suffices to count the processed message in between; here we record all message indices since the last confirmation.

The actual procedures involving i_S, i_R, Q are described in the boxed content of Fig. 9. In Init, (i_S, i_R) are both initialized to -1, the minimum message index; Q is initialized to the empty queue. In Snd, (i_R, Q) are encrypted with the sent message, and after the encryption the message index (extracted from the ciphertext c) is recorded by Q. In Rcv, $(\bar{i}_R, \bar{Q})$ are decrypted along with the message from the received ciphertext, and if the decryption succeeds (Q, i_S, i_R) are updated by running update. This update function first records the index $\bar{i}$ of the accepted message, then updates i_R when it is smaller than $\bar{i}$; next, if $i_S < \bar{i}_R$ (i.e., some of P's early actions currently recorded by Q have been known by $\bar{P}$), then it deletes those early actions and updates i_S.

Note that Ch^m remains correct since the causality-related operations (dash-boxed in Fig. 9) do not affect the input of Snd nor the output of Rcv.

SCP Security of Ch^m. Consider a function $\mathsf{localG}^{\star}_m$ that updates G_P as follows. First, it extracts the decrypted queue $\bar{Q}$ and the output index $\bar{i}$ from the input transcript T_{Rcv}. Then, it processes $\bar{Q}$ from its front (oldest) element to its back (latest) element one by one. Recall that each element e_i in $\bar{Q}$ is a message index that represents an action. Consider the i-th element e_i in $\bar{Q}$. If e_i represents a sending action, the function checks if the e_i-th sending vertex in $V_{\bar{P}}$ has been added, and if not adds it and connects it to the corresponding receiving vertex

(if any) in V_P. If e_i represents a receiving action, the function checks if the e_i-th sending vertex in V_P already connects to some receiving vertex in $V_{\bar{P}}$, and if not adds a new receiving (largest) vertex $\bar{v}$ to $V_{\bar{P}}$ and a directed edge from the e_i-th sending vertex of V_P to $\bar{v}$. After processing the entire queue $\bar{Q}$, it adds the $\bar{i}$-th sending vertex $\bar{v}'$ to $V_{\bar{P}}$ (if not yet added) and a new receiving (largest) vertex v' to V_P, then adds the edge $(\bar{v}', v')$. We illustrate the above procedures with a simple example in Fig. 10.

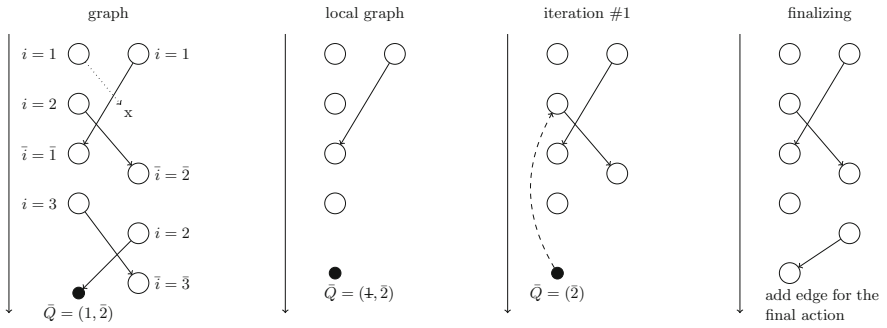

Fig. 10. Building local graph in Ch^m. The first figure shows the actual communication graph (with its first message being dropped on the network) where the left party eventually receives a ciphertext with queue $\bar{Q} = (1, \bar{2})$. Starting from its local graph (2nd figure) it iterates over the queue $\bar{Q}$, skipping the first sending vertex 1 (as it has been received) and adding the receiving vertex $\bar{2}$ as the largest vertex in the other party's vertex set and the edge (3rd figure). It finalizes the update by adding the vertices and edge of the final action (4th figure).

With $\mathsf{localG}^\star_m$, it is not hard to see that: (1) $G_P = G|_P$ always holds for a correct Ch^m execution and (2) $(G_P)_{\geq t_c+\Delta} = (G|_P)_{\geq t_c+\Delta}$ always holds for a correct Ch^m execution after recovery; we call this the correctness of $\mathsf{localG}^\star_m$. In the following theorem (with proof in the full version [9]), we show that the SCP security of the generic causal channel Ch^m can be reduced to the S-INT-CTXT and ROB-CORR security of its underlying bidirectional channel Ch.

Theorem 1. *For any $\Delta > 0$ and efficient adversary $\mathcal{A}$, there exist efficient adversaries $\mathcal{B}, \mathcal{C}$ such that*

$$\mathbf{Adv}^{\mathsf{scp}}_{\mathsf{Ch}^m,\Delta,\mathsf{localG}^\star_m}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{s\text{-}int\text{-}ctxt}}_{\mathsf{Ch},\Delta}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{rob\text{-}corr}}_{\mathsf{Ch},\Delta}(\mathcal{C}).$$

When Ch is instantiated with $\mathsf{Ch}_{\mathrm{Signal}}$, in the full version [9] we show that $\mathsf{Ch}^m_{\mathrm{cSignal}}$ provably achieves SCP security with $\Delta = 3$.

Integrating Causality in Application User Interfaces. Recall that SCP security ensures that the channel parties are in principal able to derive the correct causal information, but how to utilize it is up to the SM applications.

Here for completeness, we show a concrete method for application user interfaces to visualize causality offered by our causal channel.

Consider a message m accepted by a user, say, Alice. A causal channel can provide a causality feature that allows Alice to view which of her sent messages m depends on. To do this, the channel extracts the decrypted $\bar{Q}$ from the Rcv execution that outputs m, collects the recorded indices of messages sent by Alice, and returns those message indices along with m to the application. Then, the feature can be realized by highlighting the messages returned from the channel when Alice does a "press and hold" on the accepted message m. A toy example is described in the full version [9].

Such a causality-preserving feature helps users reduce or avoid misunderstanding caused by insufficient or incorrect causal dependencies displayed on a regular user interface (that does not preserve causality). There could be other more elegant ways to visualize causality, but finding the best visualization method and performing usability testing are beyond the scope of our work.

On the Size of Q. Recall that Q records all performed actions (as message indices) whose delivery has not yet been confirmed. From Fig. 9, we see that index queue Q dominates all overhead (computation, storage and communication). More precisely, all overhead is linear to the queue size $|Q|$. Clearly, $|Q|$ depends on the communication patterns of the conversations, for which we show two examples in the full version [9]. In practice, a straightforward way to limit such overhead is to set a *threshold* for the maximum number of elements in Q, similar to how Signal limits the maximum number of cached encrypted messages. Here, however, the causality security is slightly weakened to protect only the actions recorded in Q, for which a formal confirmation is left for future work.

6 Message Franking Channels and Causality Preservation

6.1 Message Franking Channels

In a message franking channel, besides exchanging messages the users are also allowed to report abusive messages to a third party (e.g., the messaging service provider). This additional functionality is called *message franking (MF)* by Facebook Messenger [15]. Such a setup concerns three parties: two users Alice (A), Bob (B), and a third party that we call a server (S). S routes (encrypted) messages exchanged between users (and hence S is referred to as a *router* in [19]). The role of the server is to authenticate the franking tag $c.c_f$ included in any ciphertext c routed through the server, such that the receiver (reporter) has a proof for the server to check that the other user has indeed sent that ciphertext.

A *message franking channel (MFC)* has been formalized by [19]. Similar to the discussion in Sect. 4.1, we extend their definition to capture the acting party's identity and the received index of the sending action (wrapped into the message auxiliary information), meanwhile ignoring the application-level associated data, sometimes referred to as a header. Besides, to match our bidirectional channel syntax and for better understanding, our definition is not nonce-based.

Definition 3. *A message franking channel is a five-tuple* $\mathsf{MFCh} = (\mathsf{Init}, \mathsf{Snd}, \mathsf{Rcv}, \mathsf{Tag}, \mathsf{Rprt})$ *associated with a channel key space* $\mathcal{K}_{\mathsf{Ch}}$*, a server key space* $\mathcal{K}_S$*, a state space* $\mathcal{ST}$*, a message space* $\mathcal{M}$*, an auxiliary information space* $\mathcal{U}$*, an index space* $\mathcal{I}$*, an opening key space* $\mathcal{K}_f$*, a franking tag space* $\mathcal{C}_f$*, and a tag space* $\mathcal{T}$*:*

$\mathsf{Init}(P, k) \to st_P$*: takes* $P \in \{A, B, S\}$ *and a key* k*, where* $k \in \mathcal{K}_{\mathsf{Ch}}$ *for* $P \in \{A, B\}$ *and* $k \in \mathcal{K}_S$ *for* $P = S$*, and outputs the initial state of* P*;*

$\mathsf{Snd}(P, st, m) \xrightarrow{\$} (st', c)$ *takes* $P \in \{A, B\}$*,* $st \in \mathcal{ST}$*,* $m \in \mathcal{M}$*, and outputs an updated state* $st' \in \mathcal{ST}$ *and a ciphertext* $c \in \{0,1\}^*$*, where the ciphertext contains a franking tag* $c.c_f \in \mathcal{C}_f$ *and a message index* $c.i \in \mathcal{I}$*;*

$\mathsf{Rcv}(P, st, c) \to (st', m, u, k_f)$ *takes* $P \in \{A, B\}$*,* $st \in \mathcal{ST}$*,* $c \in \{0,1\}^*$*, and outputs an updated state* $st' \in \mathcal{ST}$*, a message* $m \in \mathcal{M} \cup \{\perp\}$ *with auxiliary information* $u \in \mathcal{U}$ *that contains message index* $u.i \in \mathcal{I}$*, and an opening key* $k_f \in \mathcal{K}_f$*;*

$\mathsf{Tag}(st_S, P, c_f) \to (st'_S, \tau)$*: takes* $st_S \in \mathcal{ST}$*, (sender identity)* $P \in \{A, B\}$*,* $c_f \in \mathcal{C}_f$*, and outputs an updated state* $st'_S \in \mathcal{ST}$ *and a server tag* $\tau \in \mathcal{T}$*;*

$\mathsf{Rprt}(st_S, P, m, u, k_f, c_f, \tau) \to (st'_S, b)$ *takes* $st_S \in \mathcal{ST}$*, (reporter identity)* $P \in \{A, B\}$*,* $m \in \mathcal{M}$*,* $u \in \mathcal{U}$*,* $k_f \in \mathcal{K}_f$*,* $c_f \in \mathcal{C}_f$*,* $\tau \in \mathcal{T}$*, and outputs an updated state* $st'_S \in \mathcal{ST}$ *and a verification bit* $b \in \{0,1\}$*.*

Let $\mathsf{Ch} = (\mathsf{Init}', \mathsf{Snd}, \mathsf{Rcv}')$ *be the underlying bidirectional channel of* MFCh*, where* Init' *is* Init *with input* $P \in \{A, B\}$ *and* Rcv' *is* Rcv *with output* $(st', m, u.i)$*. Correctness requires that 1)* Ch *is correct and 2) all received messages can be* successfully reported *(i.e.,* $b = 1$*).*

A message franking channel MFCh extends its underlying bidirectional channel in several ways: (i) Init further initializes the secret state of the server; (ii) Snd and Rcv respectively further output a franking tag and an opening key used by the server to verify authenticity of user messages; (iii) Rcv outputs auxiliary information (in addition to the message index) to capture potential causality information of the received message; and (iv) Tag and Rprt are used by the server to tag encrypted messages and verify reported messages.

6.2 Causality Preservation of Message Franking Channels

As briefly explained in the introduction, there are two types of causality preservation one would expect from a message franking channel. One is security for *honest users* against *a malicious server* that acts as a network attacker, resembling our causality preservation for bidirectional channels. The other one is security for *an honest server* against *one malicious user* who knows the channel key and tries to fool the reporting system by tampering with causality.

Trust Model. Before defining security, we first clarify the trust model for message franking channels. It is usually assumed that the server-user communications are *mutually authenticated*, which in practice can be realized by, e.g.,

server-authenticated TLS connections with user login. In particular, if the server is not authenticated, a user can send abusive messages that cannot be reported; if the user is not authenticated, a user can forge and successfully report abusive messages never sent by the other user. Note that such mutual authentication guarantees message integrity against network attackers, i.e., only a malicious server is able to play man-in-the-middle attacks.

Channel Causality Preservation. First, as with bidirectional channels, we define security notions to model causality preservation for honest users, which we call *channel causality preservation (CCP)* notions. The goal of the adversary is the same as the bidirectional channel case, i.e., to make some user's local view on causality deviate from the actual case or to make some user accept a malicious message. Under our trust model, the adversary is a malicious server that mirrors a network attacker in the bidirectional channel setting.

The security experiments for both the basic and strong causality preservation of a message franking channel MFCh are defined in the same way as depicted in Fig. 3 and Fig. 4, except that the bidirectional channel algorithms $\mathsf{Init}, \mathsf{Snd}, \mathsf{Rcv}$ are replaced by those of MFCh and the message index is extracted from the accepted auxiliary information. The corresponding advantage measures $\mathbf{Adv}^{\mathsf{cp}}_{\mathsf{MFCh},\mathsf{localG}}(\mathcal{A})$ and $\mathbf{Adv}^{\mathsf{scp}}_{\mathsf{MFCh},\Delta,\mathsf{localG}}(\mathcal{A})$ are also defined in the same way. Note that the server-related algorithms $\mathsf{Tag}, \mathsf{Rprt}$ do not show up in the above security definitions because the adversary plays the role of a malicious server and knows the server secrets. One can also define the integrity notions for message franking channels as with Fig. 5 and Fig. 6 and derive similar relationship between CCP notions and integrity notions as with Fig. 7.

Report Causality Preservation. Then, we model the causality security that is directly related to the "message franking" functionality, which we call *report causality preservation (RCP)*. To define such security, it is convenient to view the adversary as either a malicious sender or a malicious receiver (reporter), like [18,19] defining *sender-binding* and *receiver-binding* notions for message franking schemes. Sender binding guarantees that no malicious user can make the other user accept a message that cannot be reported (and hence the correct causal information cannot be reported); receiver binding guarantees that no malicious user can successfully report a message that is never sent by the other user. Similarly, we split our RCP notion into two parts: RCP-S and RCP-R.

Our RCP-S notion (see Fig. 11 for its security experiment $\mathbf{Exp}^{\mathsf{rcp\text{-}s}}_{\mathsf{MFCh},\mathcal{A}}(1^\lambda)$) is *equivalent* to the sender binding notion defined in [19], except that we add a `Send` oracle to allow an honest party to send messages and our MFC syntax uses probabilistic AEAD and ignores headers. This notion is a "bidirectional channel" extension of the "unidirectional" sender-binding property defined in [18], and the adversarial goal in our model is again to make an honest user accept an unreportable message. Note that in $\mathbf{Exp}^{\mathsf{rcp\text{-}s}}_{\mathsf{MFCh},\mathcal{A}}(1^\lambda)$, the `Recv` oracle is required to process only ciphertexts with valid tags output by `Tag`, because the trust

model assumes that users can only receive messages through the server (otherwise RCP-S is easy to break). Also note that although a malicious sender can manipulate the global causality graph, once the local graph is settled on the honest receiver side, this graph is deemed correct and cannot be modified; therefore, causality-related functionality is irrelevant to the definition of RCP-S. More detailed description of RCP-S is omitted here due to its high similarity to [19]. The RCP-S adversarial advantage of a message franking channel MFCh is defined as $\mathbf{Adv}^{\mathsf{rcp\text{-}s}}_{\mathsf{MFCh}}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\mathsf{rcp\text{-}s}}_{\mathsf{MFCh},\mathcal{A}}(1^\lambda) \Rightarrow 1]$. We say MFCh is RCP-S-secure if its RCP-S advantage is negligible for any efficient adversary $\mathcal{A}$.

Our RCP-R notion (formally defined later) also follows the receiver-binding definitions [18,19], but it is extended to further allow the adversary to win if it successfully reports a message that carries *wrong or insufficient* causal information. As explained in the introduction, such information is very important for message franking because a benign message may look abusive when taken out of context. By design, RCP-R obviously implies receiver binding, which is defined as RCP-R excluding causality-related parts. Such a receiver binding notion (omitted here for conciseness) is essentially equivalent to receiver binding defined in [19]. However, the other direction is not true, i.e., receiver binding does not imply RCP-R. For instance, as shown in Sect. 7.1, Facebook's message franking channel $\mathsf{MFCh}_{\mathrm{FB}}$ does not achieve RCP-R security, but with a theorem very similar to Theorem 3 (shown in Sect. 7.2) one can prove that $\mathsf{MFCh}_{\mathrm{FB}}$ satisfies receiver binding.

We say a message franking channel *preserves report causality* (or is RCP-secure) if it is both RCP-S-secure and RCP-R-secure. In the following, we show the formal definition of our RCP-R security.

Message-Dependency Graph and its Extractor. First, we clarify what causal information is considered sufficient for a message m sent by an honest party P and reported by a malicious user $\bar{P}$. Ideally, the entire causal information until the sending action of the reported message could be carried by the m's auxiliary information, but this leads to expensive communication overhead. Instead, it suffices to carry only the causal information not yet confirmed by $\bar{P}$ in P's view, because the confirmed causal information has already been carried by the auxiliary information of messages accepted by P and hence can be reported. The above not-yet-confirmed causal information is exactly what queue Q records in the causal channel Ch^m (see Fig. 9) appended with the index i of the reported message m. We call the causality graph that represents the above causal information associated with each message the *message-dependency graph*. Let $G|^i_P$ denote the message-dependency graph of the i-th message sent by party P, which is a subgraph of $G|_P$. For instance, consider the message sent by Bob at b_6 in Fig. 2. This message has index 4 and $G|^4_B$ consists of (a_1, b_3), b_4, (a_5, b_5), and b_6, because the received message at b_5 already confirmed the delivery of messages sent at b_1 and b_2. Note that $G|^i_P$ is necessary for the server to construct the restricted causality graph $G|_P$ of the accused honest party P.

A message-dependency graph extractor Extr is a function that takes a message's auxiliary information and outputs a message-dependency graph.

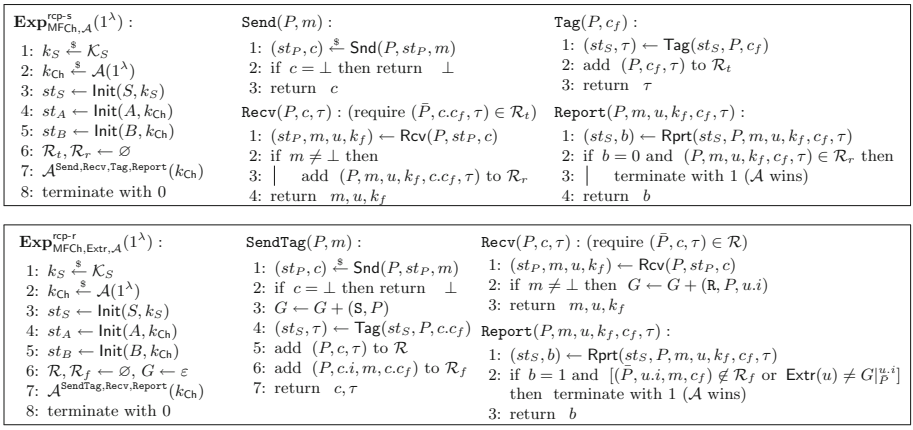

Fig. 11. Security experiments for report causality preservation

Security Experiment for RCP-R. On the bottom of Fig. 11, we depict the RCP-R security experiment $\mathbf{Exp}^{\mathsf{rcp\text{-}r}}_{\mathsf{MFCh},\mathsf{Extr},\mathcal{A}}(1^\lambda)$, which is associated with a message franking channel $\mathsf{MFCh} = (\mathsf{Init}, \mathsf{Snd}, \mathsf{Rcv}, \mathsf{Tag}, \mathsf{Rprt})$ and a message-dependency graph extractor Extr.

In the beginning, the challenger samples a random server key k_S and the adversary outputs an arbitrary channel key k_{Ch}, then the Init algorithm is executed to derive the initial states. All the states used in the game are also properly initialized. Then, $\mathcal{A}$ inputs the channel key k_{Ch} and is given access to three oracles `SendTag`, `Recv` and `Report`:

`SendTag` takes a user identity and a message, calls Snd on the input message, updates the graph, calls Tag on the franking tag (included in the derived ciphertext), records useful information in $\mathcal{R}$ and $\mathcal{R}_f$, and returns the derived ciphertext and server tag. This oracle models a user sending messages honestly through the server. Recall that malicious senders are already captured by RCP-S, whose goal is to make the other user accept unreportable messages.

`Recv` takes a user identity, a ciphertext and a server tag, calls Rcv on the input ciphertext, updates the graph, and outputs the derived message with auxiliary information and the derived opening key. Note that this oracle does not give the adversary much additional ability, because as a malicious receiver it already knows the secret user state to decrypt any ciphertext. The purpose of this oracle is to allow an honest party receive messages (through the server) and to update the global causality graph G (used to detect maliciously reported causal information). Therefore, we can require the oracle to only process ciphertexts and server tags output by `SendTag` queries.

`Report` takes a reporter (receiver) identity, a message with auxiliary information, an opening key, a franking tag, and a server tag, calls Rprt on the oracle input, and returns the derived verification bit b. The adversary wins if it reports successfully ($b = 1$) with either a message never output by an honest sender ($(\bar{P}, u.i, m, c_f) \notin \mathcal{R}_f$) or incorrect causal information ($\mathsf{Extr}(u) \neq G|^{u.i}_{\bar{P}}$).

Advantage Measure of RCP-R. The RCP-R advantage is defined as $\mathbf{Adv}^{\mathsf{rcp\text{-}r}}_{\mathsf{MFCh},\mathsf{Extr}}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\mathsf{rcp\text{-}r}}_{\mathsf{MFCh},\mathsf{Extr},\mathcal{A}}(1^\lambda) \Rightarrow 1]$ for any arbitrary extractor Extr. We say a message franking channel MFCh is RCP-R-secure if one can *construct* an efficiently computable function $\mathsf{Extr}^\star$ such that, for any efficient adversary $\mathcal{A}$, the advantage $\mathbf{Adv}^{\mathsf{rcp\text{-}r}}_{\mathsf{MFCh},\mathsf{Extr}^\star}(\mathcal{A})$ is negligible. That is, a RCP-R-secure message franking channel guarantees that the server can use $\mathsf{Extr}^\star$ to derive all causal information captured by the associated message-dependency graph of each successfully reported message.

Remark on RCP-R Security. Note that RCP-R security both guarantees the authenticity of the reported message and extends it to the message flow. The reported flow itself, however, does not include the content of previous messages but only contains information about the related causal relations (to reduce the overhead). In case of a dispute, the accused party can then report the content of the previous messages for the server to reconstruct the communication. We discuss this process in more detail for the concrete case of Facebook Messenger at the end of Sect. 7.2.

7 Causality Preservation of Facebook's Message Franking

In this section, we first describe Facebook Messenger's message franking scheme [15] and show its insecurity for preserving report causality, then amend it to provably achieve the desired security.

7.1 Facebook's Message Franking Channel and Its Insecurity

Facebook's Message Franking Channel. Following our message franking channel syntax (see Definition 6.1), we present Facebook's MFC as a message franking channel $\mathsf{MFCh}_{\mathrm{FB}}$ in Fig. 12, in a *generic* style for the benefit of modular design. That is, we abstract $\mathsf{MFCh}_{\mathrm{FB}}$ as constructed with a bidirectional channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Snd}, \mathsf{Rcv})$, a commitment scheme with verification $\mathsf{CS} = (\mathsf{Com}, \mathsf{VerC})$, and a MAC $\mathsf{MAC} = (\mathcal{K}, \mathsf{Mac}, \mathsf{Ver})$, where Facebook Messenger uses Signal as the underlying bidirectional channel protocol (i.e., $\mathsf{Ch} = \mathsf{Ch}_{\mathrm{Signal}}$) and instantiates both CS and MAC with HMAC-SHA-256 HMAC [4]. Correctness of $\mathsf{MFCh}_{\mathrm{FB}}$ follows from that of its building blocks Ch, MAC, and CS.

Causality Insecurity of $\mathsf{MFCh}_{\mathrm{FB}}$. First, as shown in Sect. 5.1, we know $\mathsf{MFCh}_{\mathrm{FB}}$ does not preserve channel causality when Ch is instantiated with

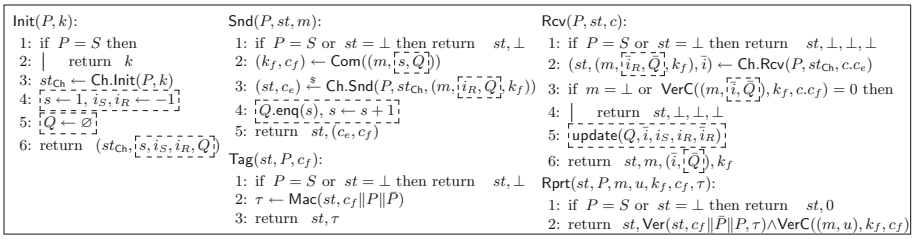

Fig. 12. Facebook's message franking channel $\mathsf{MFCh}_{\mathrm{FB}}$ (without boxed content) and the causal message franking channel $\mathsf{MFCh}_{\mathrm{cFB}}$ (with boxed content). The update function is the same as defined in Fig. 9.

$\mathsf{Ch}_{\mathrm{Signal}}$. Then, in the following we show that $\mathsf{MFCh}_{\mathrm{FB}}$ does not achieve RCP security (more specifically, RCP-R security) either, even if Ch is instantiated with our proposed causal Signal channel $\mathsf{Ch}^{m}_{\mathrm{cSignal}}$. The key observation is that the server receives only the reported message and its index, but not any other causal information. For instance, for the two execution flows considered in our Signal causality attack depicted in Fig. 8, when the message m_3 associated with c_3 is reported, the server cannot distinguish the two flows (that lead to different message-dependency graphs). That is, any extractor Extr will output an incorrect message-dependency graph associated with m_3 with probability at least $1/2$, i.e., $\mathbf{Adv}^{\mathsf{rcp\text{-}r}}_{\mathsf{MFCh}_{\mathrm{FB}},\mathsf{Extr}}(\mathcal{A}) \geq 1/2$ for any possible extractor Extr. By definition, $\mathsf{MFCh}_{\mathrm{FB}}$ does not achieve RCP-R security.

7.2 Integrating Causality in Facebook's Message Franking

The Causal Message Franking Channel. As shown in Fig. 12 with boxed content, our causal message franking channel $\mathsf{MFCh}_{\mathrm{cFB}}$ amends Facebook's message franking channel by adding a queue Q (defined in Sect. 5.2) to the auxiliary information of each sent message. This is quite similar to the Signal case, so the performance overhead introduced by $\mathsf{MFCh}_{\mathrm{cFB}}$ is also linear in $|Q|$ as discussed in Sect. 5.2. It is also easy to check that $\mathsf{MFCh}_{\mathrm{cFB}}$ remains correct.

CCP Security of $\mathsf{MFCh}_{\mathrm{cFB}}$. Consider a local graph update function $\mathsf{localG}^{\star}$ that extracts $\bar{Q}$ and $\bar{i}$ from the input transcript T_{Rcv} and proceeds as $\mathsf{localG}^{\star}_{m}$ for Ch^{m}. With a proof (omitted here) very similar to that of Theorem 1, we have the following theorem showing that the SCP security of our proposed causal message franking channel $\mathsf{MFCh}_{\mathrm{cFB}}$ can be reduced to the S-INT-CTXT and ROB-CORR security of the underlying bidirectional channel Ch.[9] In particular, the latter holds for $\Delta = 3$ when Ch is instantiated with $\mathsf{Ch}_{\mathrm{Signal}}$ (e.g., for Facebook Messenger), as discussed in the full version [9].

[9] A similar theorem (omitted here) holds for the case of basic causality preservation.

Theorem 2. *For any $\Delta > 0$ and any efficient adversary $\mathcal{A}$, there exist efficient adversaries $\mathcal{B}, \mathcal{C}$ such that*

$$\mathbf{Adv}^{\mathsf{scp}}_{\mathsf{MFCh}_{\mathrm{cFB}},\Delta,\mathsf{localG}^\star}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{s\text{-}int\text{-}ctxt}}_{\mathsf{Ch},\Delta}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{rob\text{-}corr}}_{\mathsf{Ch},\Delta}(\mathcal{C}).$$

RCP Security of $\mathsf{MFCh}_{\mathrm{cFB}}$. First, for almost the same reason why Facebook's message franking scheme satisfies perfect sender binding in [18], we can conclude that $\mathsf{MFCh}_{\mathrm{cFB}}$ achieves perfect RCP-S security (i.e., $\mathbf{Adv}^{\mathsf{rcp\text{-}s}}_{\mathsf{MFCh}_{\mathrm{cFB}}}(\mathcal{A}) = 0$). This is because `Recv` in the RCP-S security game (see top of Fig. 11) processes only ciphertexts with a *valid* server tag (i.e., sent through the server) and Rcv runs the *same* VerC check as in Rprt before accepting a message. Actually, with the same argument one can show that the original Facebook's MFC $\mathsf{MFCh}_{\mathrm{FB}}$ is also RCP-S-secure. Then, for RCP-R security, consider a message-dependency graph extractor $\mathsf{Extr}^\star$ that takes $(\bar{i}, \bar{Q})$ from the input auxiliary information u and then proceeds as $\mathsf{localG}^\star_m$ for Ch^m, but now updating an empty local graph. The following theorem (proved in the full version [9]) shows that $\mathsf{MFCh}_{\mathrm{cFB}}$ preserves report causality if its underlying MAC and CS schemes are secure. The latter holds when both instantiated with HMAC [3,18].

Theorem 3. *For any efficient adversary $\mathcal{A}$, there exist efficient adversaries $\mathcal{B}, \mathcal{C}$ such that*

$$\mathbf{Adv}^{\mathsf{rcp\text{-}r}}_{\mathsf{MFCh}_{\mathrm{cFB}},\mathsf{Extr}^\star}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{v\text{-}bind}}_{\mathsf{CS}}(\mathcal{C}).$$

Improving Dispute Handling with Causality. Here we show how causality can be utilized by a message franking server to handle disputes in a more reliable way. In particular, the $\mathsf{MFCh}_{\mathrm{cFB}}$ server can construct $\mathsf{Extr}^\star$ to extract the message-dependency graph when dealing with abuse reports. Since now the server knows how the reported message depends on previous messages (without knowing the content), the server can ask the users to report those messages for further consideration if the accused user wants to defend himself. This process can continue until the fact is clear, which is always viable because in the worst case the entire communication with the correct causal information is revealed.

For instance, consider the attack discussed in the introduction: Alice asks Bob "what was the worst insult you have ever heard?" and reports the received response. The server now gets the exact message dependencies of the reported message (which may be visualized as a causality graph or something similar) and knows that Bob indeed received some message from Alice before sending the reported message, so it can ask Bob if he wants to report that message to defend himself. In this way, the above causality attack can be prevented.

8 Conclusion

We have seen that causality in two-user messaging channels can be preserved if one transmits sufficient information on the channel to be able to reconstruct the

restricted graph. This coincides with the original idea in distributed computing to recover global states from local snapshots. It is an interesting open problem to investigate how causality can be integrated in secure *group messaging*. Another interesting problem to explore is to determine a lower bound on the time and space overhead for channels to guarantee causality security.

We remark that, from a channel perspective, we assume the *atomic* sending of messages, while for example TLS 1.3 is rather a stream-based interface [17]. Although it may seem first that our notion of causality is related only to an application-level viewpoint with atomic message processing, it is nonetheless tied to the receiving action Rcv of the channel protocol.

Finally, while not the focus of this work, it is certainly worthwhile to investigate how causality can be better visualized for users; one should also scrutinize how users respond to such designs.

Acknowledgments. We thank the anonymous reviewers for valuable comments. Shan Chen is funded by the research start-up grant by the Southern University of Science and Technology. Marc Fischlin is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - SFB 1119 - 236615297.

References

1. Alwen, J., Coretti, S., Dodis, Y.: The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. Part I, volume 11476 of LNCS, pp. 129–158. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-17653-2_5
2. Barooti, K., Collins, D., Colombo, S., Huguenin-Dumittan, L., Vaudenay, S.: On active attack detection in messaging with immediate decryption. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part IV, volume 14084 of Lecture Notes in Computer Science, pp. 362–395. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38551-3_12
3. Bellare, M.: New proofs for NMAC and HMAC: security without collision-resistance. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 602–619. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_36
4. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_1
5. Bellare, M., Kohno, T., Namprempre, C.: Authenticated encryption in SSH: provably fixing the SSH binary packet protocol. In: Atluri, V. (ed.) ACM CCS 2002, pp. 1–11. ACM Press (2002)
6. Boyd, C., Hale, B., Mjølsnes, S.F., Stebila, D.: From stateless to stateful: generic authentication and authenticated encryption constructions with application to TLS. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 55–71. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29485-8_4
7. Caforio, A., Durak, F.B., Vaudenay, S.: Beyond security and efficiency: on-demand ratcheting with security awareness. In: Garay, J.A. (ed.) PKC 2021. LNCS, vol. 12711, pp. 649–677. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75248-4_23

8. Chandy, K.M., Lamport, L.: Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. **3**(1), 63–75 (1985)
9. Chen, S., Fischlin, M.: Integrating causality in messaging channels. Cryptology ePrint Archive, Paper 2024/362 (2024). https://eprint.iacr.org/2024/362
10. Cohn-Gordon, K., Cremers, C.J.F., Garratt, L.: On post-compromise security. In: Hicks, M., Köpf, B. (eds.) CSF 2016 Computer Security Foundations Symposium, pp. 164–178. IEEE Computer Society Press (2016)
11. Cremers, C., Zhao, M.: Provably post-quantum secure messaging with strong compromise resilience and immediate decryption. Cryptology ePrint Archive, Report 2022/1481 (2022). https://eprint.iacr.org/2022/1481
12. Dodis, Y., Grubbs, P., Ristenpart, T., Woodage, J.: Fast message franking: from invisible salamanders to encryptment. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 155–186. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_6
13. Durak, F.B., Vaudenay, S.: Bidirectional asynchronous ratcheted key agreement with linear complexity. In: Attrapadung, N., Yagi, T. (eds.) IWSEC 2019. LNCS, vol. 11689, pp. 343–362. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26834-3_20
14. Eugster, P., Marson, G.A., Poettering, B.: A cryptographic look at multi-party channels. In: CSF 2018, pp. 31–45. IEEE (2018)
15. Facebook: Messenger secret conversations – technical whitepaper (2017)
16. Fischlin, M., Günther, F., Janson, C.: Robust channels: handling unreliable networks in the record layers of QUIC and DTLS 1.3. J. Cryptol. **37**(2), 9 (2024)
17. Fischlin, M., Günther, F., Marson, G.A., Paterson, K.G.: Data is a stream: security of stream-based channels. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 545–564. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_27
18. Grubbs, P., Lu, J., Ristenpart, T.: Message franking via committing authenticated encryption. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 66–97. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_3
19. Huguenin-Dumittan, L., Leontiadis, I.: A message franking channel. In: Yu, Yu., Yung, M. (eds.) Inscrypt 2021. LNCS, vol. 13007, pp. 111–128. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88323-2_6
20. Jaeger, J., Stepanovs, I.: Optimal channel security against fine-grained state compromise: the safety of messaging. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 33–62. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_2
21. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 273–293. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_17
22. Kohno, T., Palacio, A., Black, J.: Building secure cryptographic transforms, or how to encrypt and MAC. Cryptology ePrint Archive, Paper 2003/177 (2003). https://eprint.iacr.org/2003/177
23. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications (1978)
24. Marlinspike, M., Perrin, T.: The X3DH key agreement protocol (2016). https://www.signal.org/docs/specifications/x3dh/x3dh.pdf
25. Marson, G.A.: Real-World Aspects of Secure Channels: Fragmentation, Causality, and Forward Security. PhD thesis, Technische Universität (2017)

26. Marson, G.A., Poettering, B.: Security notions for bidirectional channels. IACR Trans. Symm. Cryptol. **2017**(1), 405–426 (2017)
27. Paterson, K.G., Ristenpart, T., Shrimpton, T.: Tag size *Does* matter: attacks and proofs for the TLS record protocol. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 372–389. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_20
28. Perrin, T., Marlinspike, M.: The double ratchet algorithm (2016). https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf
29. Pijnenburg, J., Poettering, B.: On secure ratcheting with immediate decryption. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022. Part III, volume 13793 of LNCS, pp. 89–118. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-22969-5_4
30. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (2018)
31. Rogaway, P., Zhang, Y.: Simplifying game-based definitions. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 3–32. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_1
32. Rösler, P., Mainka, C., Schwenk, J.: More is less: on the end-to-end security of group chats in Signal, WhatsApp, and Threema. In: EuroS&P, pp. 415–429. IEEE (2018)
33. Scarlata, M.: Post-compromise security and TLS 1.3 session resumption (2020)
34. Strom, R.E., Yemini, S.: Optimistic recovery in distributed systems. ACM Trans. Comput. Syst. **3**(3), 204–226 (1985)
35. Tyagi, N., Grubbs, P., Len, J., Miers, I., Ristenpart, T.: Asymmetric message franking: content moderation for metadata-private end-to-end encryption. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 222–250. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_8
36. Unger, N., et al.: SoK: secure messaging. In: 2015 IEEE Symposium on Security and Privacy, pp. 232–249. IEEE Computer Society Press (2015)