Faulty Group Protocols

Raúl Monroy and Graham Steel

Version 1.0, February 2007

1 Introduction

We survey attacks on group protocols found in the literature. This note is a companion to the group protocol corpus found at `http://homepages.inf.ed.ac.uk/gsteel/group-protocol-corpus/`. The most recent version of this document will always be available there.

2 Group Protocols

A *group protocol* is a protocol whereby the group participants securely exchange information in order to achieve a common goal or application, e.g. agreeing on a contract or on a session key. Groups are typically dynamic and relatively small in size, of the order of a hundred members [2]. To enable dynamic changes to group constitution, a group protocol may be defined by several sub-protocols for joining and leaving, and may appeal to a trusted third party, called the server for short.

The design of a group protocol may address several requirements, including efficiency (e.g., the crypt-algorithm used, the number of messages exchanged), security guarantee (e.g., implicit key agreement, exchange fairness) the type of channel (private, public), the group topology (e.g., ring), the communication technology (e.g., wired, wireless), the scheme used in key generation (centralised, distributed), the level of involvement of the server, if any (e.g., online or offline), the communication paradigm (e.g., RPC, connection-oriented) and trust hierarchy. Thus, a protocol may be correct in some scenarios but not in others. Similarly, an attack may be valid in some scenarios but not in others.

In what follows, we overview the several faulty group protocols from the literature. Protocols will be given by a sequence of steps, each of the form:

$$q.\ (e_1, \ldots, e_k, p_1, \ldots, p_m)\ a_s\langle c_1, \ldots, c_t\rangle \to a_r : m$$

meaning that, at step q, upon occurrence of events $e_1, \ldots, e_k$, and under the proviso that propositions $p_1, \ldots, p_m$ hold, participant a_s, after performing computations $c_1, \ldots, c_t$, sends message m to participant a_r. Each e_i may be either the reception of a (compound) message $rec(m_1, \ldots, m_j)$ or the generation of a random number in a given particular domain, e.g. $r_1 \in \mathbb{Z}_q^*$. Each p_i is either an identity testing ($x \stackrel{?}{=} y$) or a membership testing ($M_i \in \mathcal{G}$).

Steps will be collected into *rounds*, which we will enumerate using Roman numbers. In a round, one or more members make some contribution by sending a message to one specific individual or a group thereof. Let each group be of size n, for $n \in \mathbb{N}$, $n \geq 2$. We write the members of the group as M_i, $1 \leq i \leq n$. So a round may comprise at most the simultaneous broadcast of n messages.

K_{ij} denotes the distinct secret, long-term key that participants M_i and M_j (are able to) share and K_i denotes the long-term key that participant M_i shares with the server. $K_1^+, K_2^+, \ldots$ denote public keys and $K_1^-, K_2^-, \ldots$ the corresponding private keys; they are pair-wisely associated with principals $M_1, M_2, \ldots$. The symbols $r_1, r_2, \ldots$ denote nonces. We write $\{\!|m|\!\}_K$ (respectively $\sigma_K(m)$) to denote the encryption (respectively signing) of message m under key K. We use $\mathcal{H}(m)$ to denote a one-way, collision-resistant hash function. We use "," to denote message composition; so, abusing notation, we will simply write $\mathcal{H}(m_1, m_2)$ to denote the hashing of the result of concatenating m_1 and m_2.

The note is structured in sections, one for each principal application of group protocols: key agreement (§3), key management (§4), and contract signing (§5). Appendix A contains a list of the definitions of the security properties and the attacks used throughout this note, taken mainly from [2, 26, 22].

3 Group Key Agreement Protocols

3.1 Xor Protocol Failures

In this section, we overview two faulty protocols where the flaw stems from overlooking the algebraic properties of xor, namely: GKE.setup [4] and Bull and Otway [5]. The flaw in these protocols stems from basing the computation of a critical object on an operation of the form $M_1 \oplus \ldots \oplus M_k$. If the intruder gains knowledge of, say, M_k, he can easily compute $M_1 \oplus \ldots \oplus M_{k-1}$. The flaw is thus to do with an improper use of a crypt-algorithm: the crypt-algorithm is itself strong, but in calling for its use, the protocol designer has overlooked its properties. These pitfalls are collectively known as *protocol failures* [17, 25].

3.1.1 The GKE.setup Protocol

The GKE.setup protocol is part of the group key agreement suit for wireless networks [4]. It allows a set of mobile parties and a *trusted* server to agree on a session key. The remaining protocols, GKE.join and GKE.remove, aim to efficiently handle the dynamic changes in group membership. The GKE.setup protocol is shown in Figure 1. There K is an intermediate key, while SK is the intended session key. In the first round, each party sends the server his contribution to the key. In the second round, the server sends each party the intermediate key xored with a value computed directly from the corresponding party key contribution, thus enabling that party to retrieve the temporary key—each M_i extracts K from $K \oplus \mathcal{H}_1(\mathsf{c}, m_i^x)$.

The server has a pair of private and public keys, (x, α^x), where $x \in \mathbb{Z}_q^*$
Each group member has a pair of signing private and public keys, (K_i^-, K_i^+)
$\mathcal{G} = \{M_1, \ldots, M_n\}$ with index set I

i.	$(r_i \in \mathbb{Z}_q^*)$	$M_i\langle z_i = \alpha^{xr_i}\rangle \rightarrow \mathsf{Server}$	:	$\alpha^{r_i}, \sigma_{K_i^-}(\alpha^{r_i})$	$(i \in I)$
ii.	$(rec(m_i, \sigma_{K_i^-}(m_i))\ \ (i \in I))$	$\mathsf{Server}\langle \mathsf{c} = 0, K = \mathcal{H}_0(\mathsf{c}, \{m_i^x \mid i \in I\})\rangle \rightarrow M_i$	:	$\mathsf{c}, K \oplus \mathcal{H}_1(\mathsf{c}, m_i^x)$	$(i \in I)$

$$SK = \mathcal{H}(K \oplus \mathcal{H}_1(\mathsf{c}, m_i^x) \oplus \mathcal{H}_1(\mathsf{c}, z_i), M_1, \ldots, M_n, \mathsf{Server})$$

Figure 1: The GKE.setup protocol

According to [4], the GKE.setup protocol is provable secure with respect to Implicit Key Authentication (IKA). However, Nam *et al.* [20] found an attack on it, showing that it does not satisfy IKA, Perfect Forward Secrecy (PFS) or resistance to Known Key Attacks (KKA). The attack consists of two sessions. In the first one, the intruder participates as a valid group member but also collects all the messages exchanged. In the second one, the intruder is not part of the group, yet he impersonates a honest group member participating in both sessions, M_j, using a replay attack. Upon attack completion, the intruder knows the session key. The attack is as follows:

Once session 1 is complete $(\mathcal{G}, \mathsf{Spy} \in \mathcal{G})$

(1)	knows Spy K	
(2)	knows Spy $\alpha^{r_i}, \sigma_{K_i^-}(\alpha^{r_i})$ by eavesdropping	$(i \in I)$
(3)	knows Spy $K \oplus \mathcal{H}_1(\mathsf{c}, \alpha^{xr_i})$ by eavesdropping	$(i \in I)$
(4)	knows Spy $\mathcal{H}_1(\mathsf{c}, \alpha^{xr_i})$ by (1) and (3)	$(i \in I)$

Session 2 $(\mathcal{G}', M_j \in \mathcal{G} \cap \mathcal{G}')$

i.	$\vdots$			
	M_j	$\rightarrow \mathsf{Spy}(\mathsf{Server})$	:	$\alpha^{r_j'}, \sigma_{K_j^-}(\alpha^{r_j'})$
	$\mathsf{Spy}(M_j)$	$\rightarrow \mathsf{Server}$	:	$\alpha^{r_j}, \sigma_{K_j^-}(\alpha^{r_j})$
	$\vdots$			
ii.	$\vdots$			
	$\mathsf{Server}\langle \mathsf{c} = 0, K_j' = K' \oplus \mathcal{H}_1(\mathsf{c}, \alpha^{xr_j})\rangle$	$\rightarrow \mathsf{Spy}(M_j)$	:	c, K_j'
	$\vdots$			

knows Spy $SK' = K_j' \oplus \mathcal{H}_1(\mathsf{c}, \alpha^{xr_j})$

This attack is of type replay. The protocol flaw is that the server can authenticate M_j's message only partially: he does not know which protocol run M_j's message belongs to. Freshness can be attained by means of using timestamps or even a counter, but at the expense of having an explicit server memory requirement.

Alternatively, to fix the protocol, it is enough to ensure that the hashes have a distinct value on each run. Nam *et al.*'s fix proposal makes use of this observation. They suggest c should be used as a counter so that it is less likely that messages from different sessions look the same. Notice that even if the intruder knows the value of c he cannot easily forge $\mathcal{H}_1(\mathsf{c}, \alpha^{xr_j})$ unless he knows the private key of the server.

3.1.2 The Bull Otway Protocol

The Bull and Otway recursive authentication protocol [5], depicted in Figure 2, may be used to establish a chain of session keys between the participants involved in a protocol run.

$\mathcal{G} = \{M_1, \ldots, M_n\}$ with index set I			
i. $(rec(m), r_i \in \mathbb{Z}_q^*)$	$M_i \rightarrow M_{i+1}$	:	$\{\!\vert M_i, M_{i+1}, r_i, m \vert\!\}_{K_i}$ $\quad (1 \le i < n)$
$(rec(m), r_n \in \mathbb{Z}_q^*))$	$M_n \rightarrow \mathsf{Server}$	:	$\{\!\vert M_n, \mathsf{Server}, r_n, m \vert\!\}_{K_n}$
ii. $(rec(\{\!\vert M_n, \mathsf{Server}, r_n, \{\!\vert \ldots \vert\!\}_{\ldots} \vert\!\}_{K_n}))$	$\mathsf{Server} \rightarrow M_n$	:	$K_{n-1n} \oplus \mathcal{H}_{K_n}(r_n), \{\!\vert M_n, M_{n-1}, r_n \vert\!\}_{K_{n-1n}}$,
			$K_{n-1n} \oplus \mathcal{H}_{K_{n-1}}(r_{n-1}), \{\!\vert M_{n-1}, M_n, r_{n-1} \vert\!\}_{K_{n-1n}}$,
			$K_{n-2n-1} \oplus \mathcal{H}_{K_{n-1}}(r_{n-1}), \{\!\vert M_{n-1}, M_{n-2}, r_{n-1} \vert\!\}_{K_{n-2n-1}}$,
			$\vdots$
			$K_{12} \oplus \mathcal{H}_{K_1}(r_1), \{\!\vert M_1, M_2, r_1 \vert\!\}_{K_{12}}$
iii. $rec(m, m_1, m_2, \ldots, m_{2n-3})$	$M_n \rightarrow M_{n-1}$	:	$m_1, m_2, \ldots, m_{2n-3}$
$rec(m, m', m_1, \ldots, m_{2(i-1)-3})$	$M_i \rightarrow M_{i-1}$	:	$m_1, \ldots, m_{2(i-1)-3}$ $\quad (1 \le i < n)$
Remark: In the initial round, M_1 set $e_1 = rec(\epsilon)$, where ϵ represents the empty message.			

Figure 2: The Bull-Otway protocol

In this case, the problem is more dramatic since the protocol itself reveals information the designers never anticipated. In a protocol run with n participants, the server would issue a message of the form:

$$K_{12} \oplus x_1, K_{12} \oplus x_2, K_{23} \oplus x_2, \ldots, K_{n-2n-1} \oplus x_n, K_{n-1n} \oplus x_n$$

where K_{ii+1} is the session key that participants i and $i{+}1$ would use for secure communication. The designers intent was that only participant i would be able to extract K_{i-1i} and K_{ii+1} from $K_{i-1i} \oplus x_i$ and $K_{ii+1} \oplus x_i$. However, as reported in [24], for any concrete run, one can easily compute $K_{12} \oplus K_{23}, K_{23} \oplus K_{34} \ldots, K_{n-2n-1} \oplus K_{n-1n}$ simply by eavesdropping this message and then properly applying xor. The compromise of one of these session keys leads therefore to the compromise of all. Put differently, if the intruder is an insider, he will be able to work out every pair key K_{ii+1}, $(1 \le i \le n)$. To patch the Bull and Otway protocol, it is enough to ensure that the pairs of hashes have different values [24].

3.2 The Boyd-González Nieto Key Agreement Protocol

The Boyd and González Nieto protocol [3], see Figure 3, is an efficient multi-party key agreement protocol, which does not use a trusted third party. In return, the protocol uses a semi-trusted initiator but the authors did not state this explicitly. Nor did they indicate how initiator selection should proceed. The protocol was not designed to provide forward secrecy on account of favouring efficiency, but it was 'proven' secure.

$\mathcal{G} = \{M_1, \ldots, M_n\}$ with index set I		
i. $(r_1 \in \mathbb{Z}_q^*)$	$M_1 \rightarrow$ ALL :	$\mathcal{G}, \sigma_{K_1^-}(\mathcal{G}, \{\!\vert r_1 \vert\!\}_{K_2^+}, \ldots, \{\!\vert r_1 \vert\!\}_{K_n^+})$
ii.	$M_1 \rightarrow$ ALL :	$\{\!\vert r_1 \vert\!\}_{K_i^+} \quad (1 < i \le n)$
iii. $\left(\begin{array}{l} rec(\mathcal{G}, \sigma_{K_1^-}(\mathcal{G}, m_2, \ldots, m_n)) \\ rec(\{\!\vert r_1 \vert\!\}_{K_i^+}) \\ r_i \in \mathbb{Z}_q^* \\ M_i \in \mathcal{G},\ m_i \stackrel{?}{=} \{\!\vert r_1 \vert\!\}_{K_i^+} \end{array}\right)$	$M_i \rightarrow$ ALL :	$M_i, r_i \quad (1 < i \le n)$
$SK_n = \mathcal{H}(r_1, \ldots, r_n)$		

Figure 3: The Boyd-González Nieto protocol

Later on, Choo *et al.* [7] demonstrated that Boyd and González Nieto's 'proof' was flawed, showing that the protocol is subject to an unknown share key attack. The attack is of type interleaving replay; the intruder intercepts

the messages sent by the initiator of a protocol run and then poses himself as an initiator of a different session. The attack is as follows:

$$(\mathcal{G}_1 = \{M_1, M_2, M_3\}, \mathcal{G}_2 = \{\mathsf{Spy}, M_2, M_3\})$$

i.	$M_1 \to \mathsf{Spy}(\mathrm{ALL}_{\mathcal{G}_1})$	:	$\mathcal{G}_1, \sigma_{d_{M_1}}(\mathcal{G}_1, \{\!|r_1|\!\}_{e_{M_1}}, \ldots, \{\!|r_1|\!\}_{e_{M_n}})$
i′.	$\mathsf{Spy} \to \mathrm{ALL}_{\mathcal{G}_2}$	:	$\mathcal{G}_2, \sigma_{d_{\mathsf{Spy}}}(\mathcal{G}_2, \{\!|r_1|\!\}_{e_{M_1}}, \ldots, \{\!|r_1|\!\}_{e_{M_n}})$
ii.	$M_1 \to \mathsf{Spy}(\mathrm{ALL}_{\mathcal{G}_1})$	:	$\{\!|r_1|\!\}_{e_{M_i}}\ \ (1 \le i \le 3)$
ii′.	$\mathsf{Spy} \to \mathrm{ALL}_{\mathcal{G}_2}$	:	$\{\!|r_1|\!\}_{e_{M_i}}\ \ (2 \le i \le 3, i = \mathsf{Spy})$
iii.	$M_2 \to \mathrm{ALL}_{\mathcal{G}_2}$	:	M_2, r_2
	$M_3 \to \mathrm{ALL}_{\mathcal{G}_2}$	:	M_3, r_3
iii′.	$\mathsf{Spy}(M_2) \to M_1$	:	M_2, r_2
	$\mathsf{Spy}(M_3) \to M_1$	:	M_3, r_3

$$SK_{\mathcal{G}_1} = SK_{\mathcal{G}_2} = \mathcal{H}(r_1, r_2, r_3)$$

To fix this protocol, Choo *et al.* suggest to add the initiator name in each encrypted message component, $\{\!|M_1, r_1|\!\}_{e_{M_i}}$ $(1 \le i \le n)$ and add a unique session identifier, *sid*, in the key derivation function. It is standard in the literature to use the concatenation of all messages sent as a construct for *sid*.

3.3 Diffie-Hellman Key Exchange

Ateniese, Steiner and Tsudik [2] have introduced a collection of security protocols for group key agreement. The protocols are introduced in an orderly manner, in terms of the security guarantees they aim to provide. The protocols are all extensions to the Diffie-Hellman (DH) key agreement protocol [9] and are designed on the premise that it should be possible to base all the security properties of a protocol on a single hard problem, the DH decision problem in this case. Ateniese *et al.* provide proof sketches showing the protocols are provably secure.

Nevertheless, in 2001, Pereira and Quisquater proved these protocols are all faulty [23] and in a subsequent paper they proved it is impossible to design an scalable group key agreement protocol based solely on the A-GDH building blocks. In particular, they showed a systematic way of finding an attack against any A-GDH protocol with at least four participants.

The A-GDH protocols' main drawback is that their messages, and the components thereof, do not provide integrity. Without message integrity, achieving message authentication is hopeless. The problem is as follows. Let G be a finite cyclic group of prime order q, α a generator of G and p be a *safe prime*, such that $p = 2q + 1$. Let K_{ij} be the distinct secret, long-term key that agents M_i and M_j (are able to) share, allowing communication from M_i to M_j (NB. keys are unidirectional). Let $r_1 \in \mathbb{Z}_q^*$. Then, $\alpha^{r_1} \bmod p$ and $\alpha^{r_1 K_{12}} \bmod p$ are indistinguishable: they are members of the same group!

In all these protocols, when an agent receives a message, she exponentiates the components with a random key contribution, without checking its constitution or origin, and forwards the result to the next group member. Thus, to share a key with an honest member, M_i, the intruder only needs to gain knowledge of a pair (α^x, α^y) such that α^y is equal to the result of the exponentiation that M_i would apply to α^x [23].

We illustrate the protocol design flaw using Ateniese *et al.*'s strongest protocol, namely: SA-GDH.2 with key confirmation. The protocol is shown in Figure 4, where for the sake of readability we have assumed that all operations are mod p.

i.	$(rec(m_1, \ldots, m_n), r_i \in \mathbb{Z}_q^*)$	$M_i \to M_{i+1}$:	$m_1^{r_i K_{i1}}, \ldots, m_{i-1}^{r_i K_{ii-1}}, m_i, m_{i+1}^{r_i K_{ii+1}}, \ldots, m_n^{r_i K_{in}}$ $(1 \le i \le n-1)$
ii.	$(rec(m_1, \ldots, m_n), r_n \in \mathbb{Z}_q^*)$	$M_i \to$ ALL :	$m_1^{r_n K_{n1}}, \ldots, m_{n-1}^{r_n K_{nn-1}}, \alpha^{F(r_1 \ldots r_n)}$

$$SK_n = \alpha^{F(r_1 \ldots r_n)}$$

Remark: In the initial round M_1 sets $e_1 = rec(\alpha, \ldots, \alpha)$, the length of the message being n.

Figure 4: The SA-GDH.2 protocol with key confirmation

The protocol is much easier to grasp by means of a concrete run; a 3-party example run is shown below:

i.	$M_1 \to M_2$	:	$\alpha, \alpha^{r_1 K_{12}}, \alpha^{r_1 K_{13}}$
	$M_2 \to M_3$	:	$\alpha^{r_2 K_{21}}, \alpha^{r_1 K_{12}}, \alpha^{r_1 K_{13} r_2 K_{23}}$
ii.	$M_3 \to$ ALL	:	$\alpha^{r_2 K_{21} r_3 K_{31}}, \alpha^{r_1 K_{12} r_3 K_{32}}, \alpha^{F(r_1 r_2 r_3)}$

According to [2], this protocol is provable secure with respect to implicit key authentication (IKA), perfect forward secrecy (PFS) and resistance to known key attacks (KKA). Moreover, Ateniese *et al.* argue that the key confirmation component, $\alpha^{F(r_1, r_2, r_3)}$, provides entity authentication. However, Pereira and Quisquater [23] proved this result is misleading. They found an attack on SA-GDH.2 with/without key confirmation, showing that it does

not satisfy IKA when the intruder is a legitimate member of some groups. Pereira and Quisquater's attack consists of two sessions (arguably, the first session has already suffered from a disruption attack); bottom line result is the spy and M_2 share a key:

		Session 1 (n=4)	
i.	$M_1 \rightarrow \mathsf{Spy}$	:	$\alpha, \alpha^{r_1 K_{1I}}, \alpha^{r_1 K_{12}}, \alpha^{r_1 K_{13}}$
	$\mathsf{Spy} \rightarrow M_2$	:	$\alpha, \alpha^{r_1}, \alpha^{r_1 K_{12}}, \alpha^{r_1 K_{12}}$
	$M_2 \rightarrow \mathsf{Spy}(M_3)$	:	$\alpha^{r_2 K_{21}}, \alpha^{r_1 r_2 K_{2I}}, \alpha^{r_1 K_{12}}, \alpha^{r_1 r_2 K_{12} K_{23}}$
	$\mathsf{Spy}(M_2) \rightarrow M_3$	:	$\alpha^{r_2 K_{21}}, \alpha^{r_1 r_2 K_{2I}}, \alpha^{r_1 r_2 K_{12} K_{23}}, \alpha^{r_1 r_2 K_{12} K_{23}}$
ii.	$M_3 \rightarrow$ ALL	:	$\alpha^{r_2 r_3 K_{21} K_{31}}, \alpha^{r_1 r_2 r_3 K_{2I} K_{3I}}, \alpha^{r_1 r_2 r_3 K_{12} K_{23} K_{32}}, \alpha^{F(r_1 r_2 r_3 K_{12} K_{13}^{-1})}$
		Session 2 (n=3)	
i.	$M_1 \rightarrow \mathsf{Spy}(M_2)$	:	$\alpha, \alpha^{r'_1 K_{12}}, \alpha^{r'_1 K_{13}}$
	$\mathsf{Spy}(M_1) \rightarrow M_2$	:	$\alpha, \alpha^{r'_1 K_{12}}, \alpha^{r_1 r_2 r_3}$
	$M_2 \rightarrow M_3$	:	$\alpha^{r'_2 K_{21}}, \alpha^{r'_1 K_{12}}, \alpha^{r_1 r_2 r_3 r'_2 K_{23}}$
ii.	$M_3 \rightarrow$ Spy(ALL)	:	$\alpha^{r'_2 r'_3 K_{21} K_{31}}, \alpha^{r'_1 r'_3 K_{12} K_{32}}, \alpha^{F(r_1 r_2 r_3 r'_2 r'_3 K_{13}^{-1})}$
	$\mathsf{Spy}(M_3) \rightarrow$ ALL	:	$\alpha^{r'_2 r'_3 K_{21} K_{31}}, \alpha^{r_1 r_2 r_3 K_{12} K_{23} K_{32}}, \alpha^{F(r_1 r_2 r_3 r'_2 K_{23})}$

In the attack a valid message component, $\alpha^{r'_1 K_{13}}$, is replaced with $\alpha^{r_1 r_2 r_3}$, which the intruder derived from session 1, where he genuinely participated. By this means, the intruder makes the target user compute a session key with a value that the user actually makes public, $\alpha^{r_1 r_2 r_3 r'_2 K_{23}}$, c.f. step 2.

Adding a key confirmation component at the end of a protocol, $\alpha^{F(r_1, r_2, r_3)}$, seems to avoid these kinds of attacks only in a wireless protocol. Key confirmation provides entity authentication but only partially, since the session key is not authenticated until the parties prove knowledge of it by using it in a subsequent communication. This has been pointed out by Burmester [6], who also adds that successful key confirmation avoids known-key attacks but at the cost of making the key distribution systems interactive: the messages the users send each other are now dependent. Thus, to prove they posses the same key, users may execute an interactive zero-knowledge proof, as suggested by Desmedt and Burmester [8].

3.4 Ad-Hoc Networking: Asokan-Ginzboorg

Asokan and Ginzboorg proposed an application level protocol for use with Bluetooth devices, [1]. The scenario under consideration is this: a group of people are in a meeting room and want to set up a secure session amongst their Bluetooth-enabled laptops. They know and trust each other, but their computers have no shared prior knowledge and there is no trusted third party or public key infrastructure available. The protocol proceeds by assuming a short group password is chosen and displayed, e.g. on a whiteboard. The password is assumed to be susceptible to a *dictionary attack*, but the participants in the meeting then use the password to establish a secure secret key.

Asokan and Ginzboorg describe two protocols for establishing such a key in their paper, [1]. Steel and Bundy ahev analysed the first of these using CORAL, [26]. Here is a description of the first Asokan–Ginzboorg protocol (which we will hereafter refer to as simply 'the Asokan–Ginzboorg protocol'). Let the group be of size n for some arbitrary $n \in \mathbb{N}, n \geq 2$. We write the members of the group as M_i, $1 \leq i \leq n$, with M_n acting as group leader.

1.	M_n	$\rightarrow$	ALL	:	$M_n, \{\| E \|\}_P$	
2.	M_i	$\rightarrow$	M_n	:	$M_i, \{\| R_i, S_i \|\}_E$	$i = 1, \ldots, n-1$
3.	M_n	$\rightarrow$	M_i	:	$\{\| \{Sj, j = 1, \ldots, n\} \|\}_{R_i}$	$i = 1, \ldots, n-1$
4.	M_i	$\rightarrow$	M_n	:	$M_i, \{\| S_i, h(S_1, \ldots, S_n) \|\}_K$	some i

What is happening here is:

1. M_n broadcasts a message containing a fresh public key, E, encrypted under the password, P, written on the whiteboard.

2. Every other participant M_i, for $i = 1, \ldots, n-1$, sends M_n a contribution to the final key, S_i, and a fresh symmetric key, R_i, encrypted under public key E.

3. Once M_n has a response from everyone in the room, she collects together the S_i in a package along with a contribution of her own (S_n) and sends out one message to each participant, containing this package $S_1, \ldots, S_n$ encrypted under the respective symmetric key R_i.

4. One participant M_i responds to M_n with the package he just received passed through a one way hash function $h()$ and encrypted under the new group key $K = f(S_1, \ldots, S_n)$, with f a commonly known function.

Asokan and Ginzboorg argue that it is sufficient for each group member to receive confirmation that one other member knows the key: everyone except M_n receives this confirmation in step 3. M_n gets confirmation from a random member of the group in step 4. Once this message is received, the protocol designers argue, agents $M_1, \ldots, M_n$ must all have the new key $K = f(S_1, \ldots, S_n)$. The protocol has three goals: the first is to ensure that a spy eavesdropping on Bluetooth communications from outside the room cannot obtain the key. Secondly, it aims to be secure against *disruption attacks*[1] by an attacker who can add fake messages, but not block or delay messages. Thirdly, it aims by means of contributory key establishment, to prevent a group of dishonest players from restricting the key to a certain range.

Analysis with the CORAL tool resulted in the discovery of the following attacks,[26]:

$$
\begin{array}{llcll}
1. & M_1 & \rightarrow & \text{ALL} & : \{\!|\, M_1, E \,|\!\}_P \\
2. & M_2 & \rightarrow & M_1 & : M_2, \{\!|\, R_2, S_2 \,|\!\}_E \\
2. & spy_{M_3} & \rightarrow & M_1 & : M_3, \{\!|\, R_2, S_2 \,|\!\}_E \\
3. & M_1 & \rightarrow & M_2 & : \{\!|\, S_2, S_2, S_1 \,|\!\}_{R_2} \\
3. & M_1 & \rightarrow & M_3 & : \{\!|\, S_2, S_2, S_1 \,|\!\}_{R_2} \\
4. & M_2 & \rightarrow & M_1 & : M_2, \{\!|\, S_2, h(S_2, S_2, S_1) \,|\!\}_{f(S_2,S_2,S_1)}
\end{array}
$$

This is a disruption attack. The spy eavesdrops on the first message 2 sent, and then fakes a message 2 from another member of the group. This results in the protocol run ending with only two of the three person group sharing the key. Additionally, CORAL found an attack for a spy who is also the group leader:

$$
\begin{array}{llcll}
1. & spy & \rightarrow & \text{ALL} & : spy, \{\!|\, E \,|\!\}_P \\
2. & M_1 & \rightarrow & spy & : M_1, \{\!|\, R_1, S_1 \,|\!\}_E \\
2. & M_2 & \rightarrow & spy & : M_2, \{\!|\, R_2, S_2 \,|\!\}_E \\
3. & spy & \rightarrow & M_1 & : \{\!|\, S_1, S_2, S_{spy} \,|\!\}_{R_1} \\
3. & spy & \rightarrow & M_2 & : \{\!|\, S_1, S_2, S'_{spy} \,|\!\}_{R_2} \\
4. & M_1 & \rightarrow & spy & : M_1, \{\!|\, S_1, h(S_1, S_2, S_{spy}) \,|\!\}_{f(S_1,S_2,S_{spy})}
\end{array}
$$

This attack is just a standard protocol run for three participants, except that in the first message 3, the spy switches in a number of his own (S'_{spy} in the place of S_2). This means that M_1 accepts the key as $f(S_{spy}, S_1, S'_{spy})$, whereas M_2 accepts $f(S_{spy}, S_1, S_2)$, and both of these keys are known to the spy.

An improved version of the protocol that protects against these attacks is also given in [26]: we can prevent the disruption attacks by encrypting the agent identifiers in messages 1 and 2. To prevent the attack by the spy inside the room, we can require message 4 to be broadcast to all participants, so that everyone can check they have agreed on the same key.

$$
\begin{array}{llcll}
1. & M_n & \rightarrow & \text{ALL} & : \{\!|\, \boxed{M_n}\, , E \,|\!\}_P \\
2. & M_i & \rightarrow & M_n & : \{\!|\, \boxed{M_i}\, , R_i, S_i \,|\!\}_E, i = 1, \ldots, n-1 \\
3. & M_n & \rightarrow & M_i & : \{\!|\, \{Sj, j = 1, \ldots, n\} \,|\!\}_{R_i}, i = 1, \ldots, n-1 \\
4. & M_i & \rightarrow & \boxed{\text{ALL}} & : M_i, \{\!|\, S_i, h(S_1, \ldots, S_n) \,|\!\}_K, \text{some } i.
\end{array}
$$

4 Group Key Management Protocols

Group key management protocols are designed to allow a trusted server to maintain a secure key for a dynamic group of agents. Group key management is an area of increasing interest, driven by applications such as database access management, virtual conferencing, secure online broadcast, etc..

4.1 Tanaka Sato

The pull-based asynchronous rekeying framework of Tanaka and Sato,[28], which was primarily concerned with minimising the burden of key updates in terms of network traffic and processor time. Two main design features were introduced for this purpose: the first was the division of the group into subgroups, each under the management of a key distribution server (KDS). The communication between the KDSs is assumed to be not only secure but also conducted under a reliable totally ordered multicast protocol (RTOMP). Following [27], in our descriptions below, we model this by assuming that all the other KDSs instantaneously update theirs, effectively reducing the model to a single server.

[1] A disruption attack is an attack whereby a spy prevents the honest agents from completing a successful run of the protocol.

The second design feature of the protocol is that agents retain a list of keys rather than just one key. They discard an old key as invalid t units of time after having received a more up-to-date key, where t is set with respect to the delay in the network. Keys are distributed only when an agent sends a request to the server. An agent will make such a request when he wants to send a multicast message, or if he receives a message encrypted under a key he doesn't know already. In both cases, he will send a message to the server giving the ID number of the newest key he has, and the server will send back all newer keys. Only the newest key is used for multicast broadcasting.

The Tanaka-Sato protocol assumes the existence of a unicast authentication protocol that allows the server to establish an individual key (Ik) with a new member joining the group. This Ik is used to encrypt all communication between that member and the server. For conciseness, in our description below, we model the underlying authentication protocol by assuming the existence of a long-term key shared by each valid potential member of the group with the KDS. Note that the attacks described below would be effective for any initial authentication protocol.

The protocol is described by four sub-protocols, shown below:

Joining the Group

1.	M_i	$\rightarrow$	S	:	$\{\!\|\text{ join }\|\!\}_{K_{M_i}}$
2.	S	$\rightarrow$	M_i	:	$\{\!\| Ik_{M_i}, Gk(n) \|\!\}_{K_{M_i}}$

In message 1, M_i wants to join the group, so sends a join request under his long-term key K_{M_i}. The server generates a fresh individual key, Ik_Mi, and a new group key $Gk(n)$. Each group key has a unique ID number n. The new individual key and group key are sent to the joining member in message 2.

Leaving the Group

1.	M_i	$\rightarrow$	S	:	$\{\!\|\text{leave}\|\!\}_{Ik_{M_i}}$
2.	S	$\rightarrow$	M_i	:	$\{\!\|\text{ack.leave}\|\!\}_{Ik_{M_i}}$

In message 1, M_i sends a request to leave encrypted under his individual key Ik. The server acknowledges the leave in message 2, and generates a new group key. This key is not distributed though, and if another membership change occurs before a request for a key is received, it will never be distributed.

Sending a message

1.	M_i	$\rightarrow$	S	:	$\{\!\|\text{send},n\|\!\}_{Ik_{M_i}}$
2.	S	$\rightarrow$	M_i	:	$\{\!\| n', [Gk(n), \ldots, Gk(n')] \|\!\}_{Ik_{M_i}}$
3.	M_i	$\rightarrow$	ALL	:	$\{\!\|\text{message}\|\!\}_{Gk(n')}$

In message 1, agent M_i signals to the server that he would like to send a message by sending what the protocol designers call a 'sequence request' message together with the ID number of the newest key he has, n. The server checks that M_i is in the group, and then sends back a list of keys. If the current key is $Gk(n')$, then list contains $Gk(j)$ for all $js.t.n \leq j \leq n'$. If no joins or leaves have occurred since M_i last received a key, it may be that $n = n'$, but in general, all the keys that have been used in between will be sent. In message 3, agent M_i broadcasts his message to the group.

Receiving a message

1.	M_j	$\rightarrow$	S	:	$\{\!\|\text{read},n\|\!\}_{Ik_{M_j}}$
2.	S	$\rightarrow$	M_i	:	$\{\!\| n', [Gk(n), \ldots, Gk(n')] \|\!\}_{Ik_{M_i}}$

Suppose a multicast message has been broadcast, as in message 3 of the 'sending a message' fragment above. When another agent M_j receives the message, he first checks to see if he can read it with any of his keys. If not, he sends the ID of his newest key to the server, who returns a list of keys as for the send sub-protocol above.

Several attacks wer found on this protocol as a result of some formal analysis by Taghdiri and Jackson, [27]. This retention of a list of keys was shown to major security problems. The most serious attack involved members of the group accepting messages from a principal outside the group. A member of the group M_1 can simply broadcast a message from inside the group, leave, and then broadcast a message using the same key. Though the group key has been updated as a result of M_1 leaving, the other agents in the group will still accept the second message as valid, as they all have the old key.

4.2 Taghdiri/Jackson version of Tanaka/Sato

In Taghdiri and Jackson's revised version of the protocol, each agent should retain only the most recent key he has received, and upon receiving a multicast message, should contact the server to confirm that it is encrypted under the newest key. This may result in some message loss: delay in the network might mean that by the time a multicast message has been received, and a key request sent to the server, the group key has changed, but this was reckoned to be acceptable compared to the potential security breach.

Here is a description of the improved version of the protocol as described by Taghdiri and Jackson:

Joining the Group

1.	M_i	$\rightarrow$	S	:	$\{\|\ \text{join}\ \|\}_{K_{M_i}}$
2.	S	$\rightarrow$	M_i	:	$\{\| Ik_{M_i}, Gk(n) \|\}_{K_{M_i}}$

In message 1, M_i wants to join the group, so sends a join request under his long-term key K_{M_i}. The server generates a fresh individual key, $Ik_M i$, and a new group key $Gk(n)$. Each group key has a unique ID number n. The new individual key and group key are sent to the joining member in message 2.

Leaving the Group

1.	M_i	$\rightarrow$	S	:	$\{\| \text{leave} \|\}_{Ik_{M_i}}$
2.	S	$\rightarrow$	M_i	:	$\{\| \text{ack.leave} \|\}_{Ik_{M_i}}$

In message 1, M_i sends a request to leave encrypted under his individual key Ik. The server acknowledges the leave in message 2, and generates a new group key. This key is not distributed though, and if another membership change occurs before a request for a key is received, it will never be distributed.

Sending a message

1.	M_i	$\rightarrow$	S	:	$\{\| \text{send}, n \|\}_{Ik_{M_i}}$
2.	S	$\rightarrow$	M_i	:	$\{\| n', Gk(n') \|\}_{Ik_{M_i}}$
3.	M_i	$\rightarrow$	ALL	:	$\{\| \text{message} \|\}_{Gk(n')}$

In message 1, agent M_i signals to the server that he would like to send a message by sending what the protocol designers call a 'sequence request' message together with the ID number of the newest key he has, n. The server checks that M_i is in the group, and then sends back the newest key $Gk(n')$. If no joins or leaves have occurred since M_i last received a key, it may be that $n = n'$, but this will not be the case in general. In message 3, agent M_i broadcasts his message to the group.

Receiving a message

1.	M_j	$\rightarrow$	S	:	$\{\| \text{read}, n \|\}_{Ik_{M_j}}$
2.	S	$\rightarrow$	M_j	:	$\{\| Gk(n') \|\}_{Ik_{M_j}}$

Suppose a multicast message has been broadcast, as in message 3 of the 'sending a message' fragment above. When another agent M_j receives the message, he first sends a request to the server for the newest key. He then receives the newest key $Gk(n')$, and will only accept the multicast message if it was encrypted under that key.

One major oversight of the Taghdiri–Jackson analysis was the lack of an active intruder in their model. An active intruder of some kind has been assumed since the very first security protocol paper, [21]. His behaviour was formalised by Dolev and Yao, [10], and since then it has generally been accepted that the spy should also be able to pose as a legitimate agent, for example in Lowe's famous attack, [15]. If anything, it would seem even more likely that a multicast protocol would be subject to attack by an active intruder compared to a unicast protocol, as argued in [16]. There are inherently more opportunities for interception of traffic, and the 'crowd' of principals would typically make it easier for an intruder to pose as another legitimate principal. Such protocols should therefore be subjected to analysis under the full Dolev-Yao attacker model, as is standard for unicast protocols.

In [26], Steel and Bundy used the CORAL tool to analyse the Taghdiri–Jackson version of the protocol. This resulted in two attacks. The first is a counterexample to the *multicast group authenticity* property, 'outsider can't read', which states that non-members of the group should not be able to read group messages.

1.	spy	$\rightarrow$	server	:	$\{\| spy \|\}_{longtermK(spy)}$
2.	server	$\rightarrow$	spy	:	$\{\| Ik(1), Gk(1) \|\}_{longtermK(spy)}$
3.	M_1	$\rightarrow$	server	:	$\{\| M_1 \|\}_{longtermK(M_1)}$
4.	server	$\rightarrow$	M_1	:	$\{\| Ik(3), Gk(2) \|\}_{longtermK(M_1)}$
5.	spy	$\rightarrow$	server	:	$\{\| send(1) \|\}_{Ik(1)}$
6.	server	$\rightarrow$	spy	:	$\{\| Gk(2), send(1) \|\}_{Ik(1)}$
7.	M_1	$\rightarrow$	server	:	$\{\| send(2) \|\}_{Ik(3)}$
8.	server	$\rightarrow$	M_1	:	$\{\| Gk(2), send(2) \|\}_{Ik(3)}$
9.	M_1	$\rightarrow$	all	:	$\{\| hello(9) \|\}_{Gk(2)}$
10.	spy	$\rightarrow$	server	:	$\{\| leave \|\}_{Ik(1)}$
11.	server	$\rightarrow$	spy	:	$\{\| ackleave \|\}_{Ik(1)}$
12.	M_1	$\rightarrow$	server	:	$\{\| send(2) \|\}_{Ik(3)}$
13.	spy	$\rightarrow$	M_1	:	$\{\| Gk(2), send(2) \|\}_{Ik(3)}$
14.	M_1	$\rightarrow$	all	:	$\{\| hello(14) \|\}_{Gk(2)}$

This is an attack on the protocol which hinges on the spy sending a replayed key update message in message 13. Since the key may or may not have changed since she last saw it, agent M_1 will accept this key. The problem is that there is minimal freshness information sent in the request for a key (just the sequence number of the key an agent currently holds). Enclosing a fresh nonce inside the package sent to the server requesting a key update would blunt this attack.

CORAL also discovered and attack on the 'outsider can't send' prpoerty:

1.	M_1	$\rightarrow$	server	:	$\{\| M_1 \|\}_{longtermK(M_1)}$
2.	server	$\rightarrow$	M_1	:	$\{\| Ik(1), Gk(1) \|\}_{longtermK(M_1)}$
3.	spy	$\rightarrow$	server	:	$\{\| spy \|\}_{longtermK(spy)}$
4.	server	$\rightarrow$	spy	:	$\{\| Ik(3), Gk(2) \|\}_{longtermK(spy)}$
5.	spy	$\rightarrow$	server	:	$\{\| read \|\}_{Ik(3)}$
6.	server	$\rightarrow$	spy	:	$\{\| Gk(2) \|\}_{Ik(3)}$
7.	M_1	$\rightarrow$	server	:	$\{\| read \|\}_{Ik(1)}$
8.	server	$\rightarrow$	M_1	:	$\{\| Gk(2) \|\}_{Ik(1)}$
9.	spy	$\rightarrow$	server	:	$\{\| leave \|\}_{Ik(3)}$
10.	server	$\rightarrow$	spy	:	$\{\| ackleave \|\}_{Ik(3)}$
11.	spy	$\rightarrow$	all	:	$\{\| hello(12) \|\}_{Gk(2)}$
12.	M_1	$\rightarrow$	server	:	$\{\| read \|\}_{Ik(1)}$
13.	spy	$\rightarrow$	M_1	:	$\{\| Gk(2) \|\}_{Ik(1)}$

Like the previous attack, this is also a replay attack, in this case with the spy replaying message 8 in message 13, tricking agent M_1 into thinking that message 11 came from a legitimate member of the group. These replay attacks are more serious than those typically considered for fixed party protocols. A replay attack on a standard unicast protocol usually assumes that it would be possible for a spy to obtain a short-term key by cryptanalysis or some other means, and so it would constitute an attack on the protocol if he was able to force an agent to accept an old key. In the two attacks above, no cryptanalysis in necessary, since the spy can obtain some old keys by joining the group legitimately, and then leave before effecting the attack. However, even if we assume the spy does not have access to a valid long-term key, and so cannot join the group, these replay attacks are still dangerous. If the spy obtains a short-term group key by cryptanalysis, he can effect an attack without joining the group.

This attack can also be prevented by adding a fresh nonce to the request for a key, this time for reading a message, and including it in the reply from the server.

4.3 Iolus Protocol

The main difference between the Iolus protocol, [16], and the Taghdiri-Jackson version of the Tanaka-Sato protocol is that Iolus eagerly distributes new keys, whereas Tanaka-Sato distributes keys only on demand. Here is the protocol:

Joining the Group

1.	M_i	$\rightarrow$	S	:	$\{\| \text{join} \|\}_{K_{M_i}}$
2.	S	$\rightarrow$	M_i	:	$\{\| Ik_{M_i}, Gk(n') \|\}_{K_{M_i}}$
3.	S	$\rightarrow$	ALL	:	$\{\| Gk(n') \|\}_{Gk(n)}$

Members join the Iolus protocol in the same way as for Tanaka-Sato, i.e. by use of a pairwise authentication protocol that we model with the use of a long-term key. The server generates a fresh individual key, $Ik_M i$, and a new group key with ID n', $Gk(n')$. In message 2, the group key is sent to the new member, and in message 3, it is sent to the old members of the group under the old group key, $Gk(n)$.

Leaving the Group

1.	M_i	$\rightarrow$	S	:	$\{\| \text{leave} \|\}_{Ik_{M_i}}$
2.	S	$\rightarrow$	ALL	:	$[\{\| Gk(n') \|\}_{Ik_{M_j}} \ldots]\ \forall j \neq i, M_j \in \text{group}$

When a member M_i leaves, a new key $Gk_{n'}$ is generated and sent to each member in the form of a broadcast list. The list contains the new group key encrypted under the pairwise session key of each member still in the group (the key cannot be broadcast under the old group key, because this would give it away to the leaving member).

Sending a message

1.	M_i	$\rightarrow$	ALL	:	$\{\| \text{message} \|\}_{Gk(n)}$

The message is simply broadcast under the current group key.

In [26], Steel and Bundy report the following attack, found by CORAL. It is a counterexample to a multicast group authenticity property: when a key update to key Gk is accepted by a member of a group, and that group does not contain the spy, then the spy should not know the key Gk

1.	M_1	$\rightarrow$	server	:	$\{\!\| M_1 \|\!\}_{longtermK(M_1)}$
2.	server	$\rightarrow$	all	:	$\{\!\| Gk(1) \|\!\}_{Gk(X8)}$
3.	server	$\rightarrow$	M_1	:	$\{\!\| Ik(2), Gk(1) \|\!\}_{longtermK(M_1)}$
4.	spy	$\rightarrow$	server	:	$\{\!\| spy \|\!\}_{longtermK(spy)}$
5.	server	$\rightarrow$	all	:	$\{\!\| Gk(2) \|\!\}_{Gk(1)}$
6.	server	$\rightarrow$	spy	:	$\{\!\| Ik(5), Gk(2) \|\!\}_{longtermK(spy)}$
7.	M_2	$\rightarrow$	server	:	$\{\!\| M_2 \|\!\}_{longtermK(M_2)}$
8.	server	$\rightarrow$	all	:	$\{\!\| Gk(3) \|\!\}_{Gk(2)}$
9.	server	$\rightarrow$	M_2	:	$\{\!\| Ik(8), Gk(3) \|\!\}_{longtermK(M_2)}$
10.	M_1	$\rightarrow$	server	:	$\{\!\| leave \|\!\}_{Ik(2)}$
11.	server	$\rightarrow$	all	:	$[\{\!\| Gk(4) \|\!\}_{Ik(8)}, \{\!\| Gk(4) \|\!\}_{Ik(5)}]$
12.	spy	$\rightarrow$	server	:	$\{\!\| leave \|\!\}_{Ik(5)}$
13.	server	$\rightarrow$	all	:	$[\{\!\| Gk(5) \|\!\}_{Ik(8)}]$
14.	spy	$\rightarrow$	all	:	$[\{\!\| Gk(4) \|\!\}_{Ik(8)}, \{\!\| Gk(4) \|\!\}_{Ik(5)}]$

Again this is a replay attack. In message 14, the spy replays a key update originally sent in message 11, while he was still in the group. So, the spy knows the group key $Gk(4)$ even though he is no longer a group member. Note for this attack to work, it is necessary for two honest agents to join the group as well as the spy, whereas only one was required for the attacks in §4.2, and note further that CORAL has discovered this for itself - there is no pre-setting of the number of agents. Preventing this attack is not as straightforward as it was for the Tanaka-Sato/Taghdiri-Jackson protocol because the key updates are unsolicited, so there is no opportunity for the agents and the server to exchange a nonce. The only way to protect against replays would seem to be to include a timestamp inside the encrypted packages sent in key updates, both when agents join and leave the group. This would require all group members to have at least loosely synchronised clocks, and would further require a decision in advance about the lifetime of group keys, and the expected amount of delay in the network. However, this limitation seems inescapable for such protocols. In the presence of a Dolev-Yao spy it is insufficient, for example, to include the last key in the key update message as freshness information, since the spy may prevent this message from being received, and then re-send it once he has left the group.

5 Contract Signing Protocols

5.1 The Franklin-Tsudik Exchange Protocol

The Franklin-Tsudik protocol is an online multi-party exchange protocol where the exchange's topology is a ring. Each participant M_i offers participant M_{i+1} a message, m_i, in return of message m_{i-1}, offered by participant M_{i-1}. Subscripts are to be read mod n, where n is the number of participants in a given protocol run.

The protocol has two phases, the first of which, setup, is not specified but it is assumed that at the end of it:

- each participant knows the identity of the remaining participants of the protocol run;
- the participants have all agreed on the identitity of the server that will assist the exchange;
- the participants have all agreed on the value of the functions f and F_n that will be used. f and F_n are respectively an homomorphic one-way function and an n-variable function such that:

$$F_n(x_1, f(x_2), \ldots, f(x_n)) = f(x_1 x_2 \cdots x_n)$$

  where product is represented via juxtaposition. In [12], Franklin and Tsudik suggest to set $f(x) = x^2$ and $F_n(x_1, \ldots, x_n) = x_1^2 x_2 \cdots x_n$. And
- the descriptions of the messages to be exchanged, $f(m_i)$ $(i \in \{1, \ldots, n\})$, are made public.

The protocol is shown in Figure 5. In the first round, participants exchange promises of exchange commitment; in the second one, they send the server values from which each participant can extract the message item he is after; in the final round, after performing some checks, the server distributes these values to the participants and further information for the item extraction process. The protocol does not use any crypt-algoirthm for it is assumed to operate on private and authenticated channels.

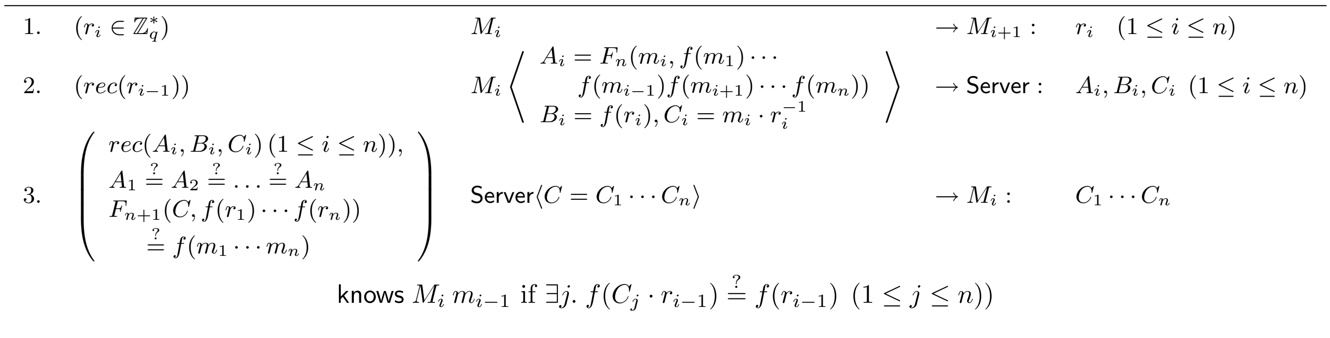

Figure 5: The Franklin-Tsudik protocol

The protocol is faulty even for $n = 2$, as illustrated in the attack below. One flaw in the protocol is that the server cannot guarantee that each participant, M_i, sent the right promise to his correspondant agent, M_{i+1} (c.f. the second message of the first round below). Exploiting this protocol flaw, an intruder can exclude an honest participant from a run. González-Deleito and Markowitch [14] suggested to fix this protocol flaw by modifying the second round messages so that each participant sends the description of his item, $f(r_i)$, as well as the description of the item he received in the first round, $f(r_{i-1})$.

1.	$M_1 \rightarrow$ Spy	:	r_1
	Spy $\rightarrow M_1$	:	r_2'
2.	$M_1 \rightarrow$ Server	:	$F(r_1, f(r_2)), f(r_1), m_1 \cdot r_1^{-1}$
	Spy $\rightarrow$ Server	:	$F(r_2, f(r_1)), f(r_2), m_2 \cdot r_2^{-1}$
3.	Server $\rightarrow M_1$	:	$m_1 \cdot r_1^{-1} \cdot m_2 \cdot r_2^{-1}$
	Server $\rightarrow$ Spy	:	$m_1 \cdot r_1^{-1} \cdot m_2 \cdot r_2^{-1}$

$$\neg(\text{knows } M_1\ m_2), \text{knows Spy } m_1$$

Yet, this check is not enough. Mukhamedov *et al.* [18] proved that the revised protocol was still faulty by showing how two non-contiguous, dishonest agents can collude to exclude two or more honest participants. So another flaw in the protocol is that the server cannot guarantee consistency on the information from which the expected message is to be extracted, $C_i = m_i r_i^{-1}$. If two non-contiguous agents, M_i and M_k, colluded one another to respectively send $C_i = m_i m_k$ and $C_k = r_i^{-1} r_k^{-1}$ (as opposed to $C_i = m_i r_i^{-1}$ and $C_k = m_k r_k^{-1}$), the server consistency test, $F_{n+1}(C, f(r_1) \cdots f(r_n)) \stackrel{?}{=} f(m_1 \cdots m_n)$, would still hold. As a result, both M_{i+1} and M_{k+1} would be left in an unfair situation. Mukhamedov *et al.* suggested to fix the revised protocol by having the server to further test that:

$$\forall i.\ \exists C_j \in \{C_1, \ldots, C_n\}.\ f(C_j \cdot r_{i-1}) = f(m_i)$$

Unfortunately, this fix throws out the baby with the bath water, because the server would be able to know each m_i $(1 \leq i \leq n)$, which he should not!

5.2 The Garay-MacKenzie Protocol

The Garay-MacKenzie (GM) protocol [13] is an optimistic, asynchronous multi-party contract signing, desgined to be fair, abuse-free and complete. The protocol involves the use of Private Contract Signatures (PFS), a cryptographic primitive with which a promise can be verified. If a party receives enough PFSs, he may use them, together with his own PFS, in order to build a verifiable signature on the contract. If problems occur, participants may run protocols to either abort or resolve the contract signing, depending on the current state of the protocol run.

The protocol is specified recursively. The description is complex enough to include it herein and so we only online it. Let n be the size of the group and let m the contract on which a signature is sought. The protocol description for participant $j < n$ is as follows. Wait for 1-level premises from M_k $(j < k \leq n)$ on m. If you do not receive them in a timely manner, quit. Otherwise, start recursive level i by sending 1-level promises on m to M_i $(1 \leq i < j)$. Wait for $(i-1)$-level promises on m from M_i $(1 \leq i < j)$, indicating that the recursive level $i - 1$ has finished. If you do not timely receive them, run the *abort* protocol. Otherwise, send i-level promises on m to M_i $(1 \leq i < j)$. Wait for i-level promises on m from M_i $(1 \leq i < j)$, indicating that the recursive level i is complete. If you do not timely receive them, run the *recovery* protocol. Otherwise, recursively complete all higher recursive levels, by sending j level promises to M_k $(j < k \leq n)$. This process is repeated until all higher

levels are complete. A concrete run for a group of size three is shown below.[2] Notice that for a group of size n, the participants exchange promises of level i, for $1 \leq i \leq n+1$. Also notice that the protocol is unbalanced: M_i participates more times than M_k, for $i < k$.

1.	$M_3 \rightarrow M_1$	: 1	8.	$M_2 \rightarrow M_1$	: 3
	$M_3 \rightarrow M_2$	: 1	9.	$M_1 \rightarrow M_2$	: 3
2.	$M_2 \rightarrow M_1$	: 1		$M_1 \rightarrow M_3$	: 3
3.	$M_1 \rightarrow M_2$	: 1	10.	$M_2 \rightarrow M_3$	: 3
4.	$M_2 \rightarrow M_1$	: 2	11.	$M_3 \rightarrow M_1$	: 4
5.	$M_1 \rightarrow M_2$	: 2		$M_3 \rightarrow M_2$	: 4
	$M_1 \rightarrow M_3$	: 2	12.	$M_2 \rightarrow M_3$	: 4
6.	$M_2 \rightarrow M_3$	: 2		$M_2 \rightarrow M_1$	: 4
7.	$M_3 \rightarrow M_2$	: 3	13.	$M_1 \rightarrow M_2$	: 4
	$M_3 \rightarrow M_1$	: 3		$M_1 \rightarrow M_3$	: 4

The abort protocol works as follows: assume the server is contacted with a request to abort. If it is the first request or if the protocol has not already been recovered, the server will accept the request, sending an abort token and updating its view of the protocol run. Conversely, if the protocol has already been successfully recovered, the server will deny it, sending a signed contract and updating its view of the protocol run.

The recovery protocol works as follows: assume the server is contacted with a request to recover. If it is the first request or if the protocol has already been recovered, then the server will accept the request, sending a signed contract and updating its view of the protocol run. Conversely, if the protocol has already been aborted, then the server must decide whether to maintain the abort or to overturn it, aiming to maintain fairness.

The GM protocol is faulty. Chadha *et al.* found several attacks for a group of size four. The attacks all follow the outline below [13]:

1. At the beginning of the protocol run, a dishonest participant, M_k, contacts the server with a request to abort, which the server accepts. The dishonest participant then gets an abort token but dishonestly continues participating in the protocol run.

2. A second dishonest signer, M_i ($i < k$) then sends a request to recover at some later point. He does not succeed but manages to put any honest signer, M_j ($i < j < k$), of a higher hierarchy, up in a position from which M_j cannot recover. He also dishonestly continues the main protocol.

3. At this point, M_k and another dishonest participant, M_l ($k < l$) stop sending any further messages. So the honest participant is forced to contact the server with a recover request. M_j is not successful in making the server overturn his abort decision.

4. Then the third dishonest participant, M_l, contacts the server with a request to recover. Being at the top of the hierachy, he manages to make the server overturn the decision. Hence, while the honest participant does not get any signed contract, the honest signer's contract is obtained.

Chadha *et al.* suggested a fix to the GM protocol. While the main protocol remained the same, the recovery protocol suffered major changes. Chadha *et al.* noticed that the recovery protocol did not properly handle decision overturning. They changed the recovery protocol so that the server could infer dishonesty on the part of a party, according to the party requests. Chadha *et al.* used MOCHA, a model checker, to analyse the protocol for groups of size $n \leq 4$.

Two years later, however, Mukhamedov and Ryan [19] proved that Chadha *et al.*'s fix was still faulty. They found an attack on the revised protocol for a group of size $n > 4$. This attack was generalised to show that the message exchange structure of the GM protocol is flawed, meaning that regardless of the server's action, unfairness for a participant will always occur.

5.3 The FPH Protocol

The Ferrer, Payeras and Huguet (FPH) protocol [11] is an optimistic, asynchronous multi-party contract signing. The group topology is a ring; the contract itself, m, specifies the order of the participants in the ring. To prove to an external party the existance of a contract signed by all parties, each participant, M_i, must obtain the signature of all the other participants on the contract, h_j, as well as their acknowledgement, ACK_j, for $1 \leq j \leq n$ and $j \neq i$. These two bits of information are exchanged in two rounds. The protocol is specified via a concrete run for three participants (see Figure fig:fph). M_1 is the first principal, M_n is the last one, and M_i ($1 < i < n$) describes the behaviour of any other intermediate participant. Notice that the last participant receives an additional acknowledgement indicating the end of the protocol run.

[2]For the sake of readability, we only write the level of the promise on each message.

1.	$M_1 \rightarrow M_i$	:	m, h_1
2.	$M_i \rightarrow M_n$	:	m, h_1, h_i
3.	$M_n \rightarrow M_1$	:	h_i, h_n
4.	$M_1 \rightarrow M_i$	:	h_n, ACK_1
5.	$M_i \rightarrow M_n$	:	ACK_1, ACK_i
6.	$M_n \rightarrow M_1$	:	ACK_i, ACK_n
7.	$M_1 \rightarrow M_i$	:	ACK_n
8.	$M_i \rightarrow M_n$	:	$ACK2$

Figure 6: The FPH protocol

Every participant may contact the server if he does not receive an expected message. This may happen exclusively at two specific points: i) the participant has sent his signature on the contract but he has not received his mates' counterpart, and ii) the participant has sent his acknowledgement but he has not received his mates' counterparts. In the former case, the participant attempts to cancel and in the latter, he attempts to recover.

Similar to the GM protocol, the FPH protocol server does not properly handle requests to cancel or recovery. Onieva *et al.* [22] found an attack on this protocol for a group of size $n > 3$, showing that the server cannot cope with a group of three contiguous dishonest participants, colluding one another to cheat on an honest participant. The attack, as introduced by [22], is shown below:

1.	$M_1 \rightarrow M_1$	:	m, h_1
2.	$M_2 \rightarrow M_3$	:	m, h_1, h_2
3.	$M_3 \rightarrow M_4$	:	m, h_1, h_2, h_3
4.	$M_4 \rightarrow M_1$	:	h_2, h_3, h_4
5.	$M_1 \rightarrow M_2$	:	h_3, h_4, ACK_1
			M_2 cancels but continues the protocol
6.	$M_2 \rightarrow M_3$	:	h_4, ACK_1, ACK_2
7.	$M_3 \rightarrow M_4$	:	ACK_1, ACK_2, ACK_3
8.	$M_4 \rightarrow M_1$	:	ACK_2, ACK_3, ACK_4
			M_1 cancels but continues the protocol
9.	$M_1 \rightarrow M_2$	:	ACK_3, ACK_4
10.	$M_2 \rightarrow M_3$	:	ACK_4
			M_3 recovers, gets the contract and continues the protocol
11.	$M_3 \rightarrow M_4$	:	$ACK2$
			M_4 attempts to recover but fails

No protocol repair is suggested in [22].

References

[1] N. Asokan and P. Ginzboorg. Key-agreement in ad-hoc networks. *Computer Communications*, 23(17):1627–1637, 2000.

[2] G. Ateniese, M. Steiner, and G. Tsudik. New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4):628–639, 2000.

[3] C. Boyd and J.-M. González Nieto. Round-optimal contributory conference key agreement. In Yvo Desmedt, editor, *Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography, PKC'03*, volume 2567 of *Lecture Notes in Computer Science*, pages 161–174. Springer, 2003.

[4] E. Bresson, O. Chevassut, A. Essiari, and D. Pointcheval. Mutual authentication and group key agreement for low-power mobile devices. *Computer Communications*, 27(17):1730–1737, 2004. A preliminary version appeared in *Proc. of the 5th IFIP-TC6/IEEE International Conference on Mobile and Wireless Communications Networks, MWCN'03.*

[5] J. Bull and D. Otway. The authentication protocol. Technical Report DRA/CIS3/PROJ/CORBA/SC/1-/CSM/436-04/0.5b, DERA, Malvern, UK, 1997.

[6] Mike Burmester. On the risk of opening distributed keys. In Yvo Desmedt, editor, *Advances in Cryptology — CRYPTO'94*, Lecture Notes in Computer Science, pages 308–317. Springer-Verlag, 1994.

[7] K.-K.-R. Choo, C. Boyd, and Y. Hitchcock. Errors in computational complexity proofs for protocols. In Bimal Roy, editor, *Advances in Cryptology - Asiacrypt 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 624–643. Springer-Verlag, 2005.

[8] Y. Desmedt and M. Burmester. Towards practical 'proven secure' authenticated key distribution. In *Proceedings of the 1st ACM conference on Computer and communications security CCS'93*, pages 228–231. ACM Press, 1993.

[9] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[10] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions in Information Theory*, 2(29):198–208, March 1983.

[11] J.-L. Ferrer-Gomila, M. Payeras-Capellà, and L. Huguet i Rotger. Efficient optimistic n-party contract signing protocol. In G.-I. Davida and Y. Frankel, editors, *Proceedings of the 4th International Conference on Information Security, ISC'01*, volume 2200 of *Lecture Notes in Computer Science*, pages 394–407. Springer, 2001.

[12] M.-K. Franklin and G. Tsudik. Secure group barter: Multi-party fair exchange with semi-trusted neutral parties. In Rafael Hirschfeld, editor, *Proceedings of the Second International Conference on Financial Cryptography, FC'98*, volume 1465 of *Lecture Notes in Computer Science*, pages 90–102. Springer, 1998.

[13] J.-A. Garay and P.-D. MacKenzie. Abuse-free multi-party contract signing. In Prasad Jayanti, editor, *Proceedings of the 13th International Symposium on Distributed Computing, DISC'99*, volume 1693 of *Lecture Notes in Computer Science*, pages 151–165. Springer, 1999.

[14] N. González-Deleito and O. Markowitch. Exclusion-freeness in multi-party exchange protocols. In *Proceedings of the 5th International Conference on Information Security, ISC'02*, volume 2433 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 2002.

[15] G. Lowe. Breaking and fixing the Needham Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055, pages 147–166. Springer Verlag, 1996.

[16] S. Mittra. Iolus: A framework for scalable secure multicasting. In *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 277–288, Cannes, France, September 1997.

[17] J.H. Moore. Protocol failures in cryptosystems. *Proceedings of the IEEE*, 76(5):594–602, 1988. Reprinted in, G.J.-Simmons, editor, *Contemporary Cryptology: The Science of Information Integrity*. IEEE Computer Science Press, 1991, pp. 541-558.

[18] A. Mukhamedov, S. Kremer, and E. Ritter. Analysis of a multi-party fair exchange protocol and formal proof of correctness in the strand space model. In A.-S. Patrick and M. Yung, editors, *Proceedings of the 9th International Conference on Financial Cryptography and Data Security, FC'05*, volume 3570 of *Lecture Notes in Computer Science*, pages 255–269. Springer, 2005.

[19] A. Mukhamedov and M.-D. Ryan. Resolve-impossibility contract for a contract-signing protocol. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop, CSFW'06*, pages 1–7. IEEE Computer Society, 2006.

[20] J. Nam, S. Kim, and D. Won. A weakness in the Bresson-Chevassut-Essiari-Pointcheval's group key agreement scheme for low-power mobile devices. *IEEE Communications Letters*, 9(5):429–431, 2005.

[21] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[22] J.-A. Onieva, J. Zhou, and J. Lopez. Attacking an asynchronous multi-party contract signing protocol. In S. Maitra, C.-E. Veni-Madhavan, and R. Venkatesan, editors, *Proceedings of the 6th International Conference on Progress in Cryptology Cryptology in India, INDOCRYPT'05*, volume 3797 of *Lecture Notes in Computer Science*, pages 311–321. Springer, 2005.

[23] O. Pereira and J.-J. Quisquater. Some attacks upon authenticated group key agreement protocols. *Journal of Computer Security*, 11(4):555–580, 2003. Special Issue: 14th IEEE Computer Security Foundations Workshop, CSFW'01.

[24] P.Y.-A. Ryan and S.-A. Schneider. An attack on a recursive authentication protocol. a cautionary tale. *Information Processing Letters*, 65(1):7–10, 1998.

[25] G. H. Simmons. Cryptanalysis and protocol failures. *Communications of the ACM*, 37(11):56–65, 1994.

[26] G. Steel and A. Bundy. Attacking group protocols by refuting incorrect inductive conjectures. *Automated Reasoning*, pages 1–28, 2005. Special Issue on Automated Reasoning for Security Protocol Analysis.

[27] M. Taghdiri and D. Jackson. A lightweight formal analysis of a multicast key management scheme. In *Proceedings of Formal Techniques of Networked and Distributed Systems - FORTE 2003*, LNCS, pages 240–256, Berlin, 2003. Springer.

[28] S. Tanaka and F. Sato. A key distribution and rekeying framework with totally ordered multicast protocols. In *Proceedings of the 15th International Conference on Information Networking*, pages 831–838, 2001.

A Glossary

A.1 Group Key Agreement Protocols

Let P be an n-party key agreement protocol, M be the set of parties and S_n be a secret jointly generated by each $M_i \in M$ as a result of running P. Then:

Definition 1 *P is* contributory *if each party equally contributes to the key and guarantee its freshness.*

All the protocols to be studied here are contributory.

Definition 2 (IKA) *P provides* implicit key authentication (IKA) *if each $M_i \in M$ is assured that no principal outside the group, $M_q \notin M$, can learn the key S_n (unless aided by a dishonest $M_j \in M$).*

IKA does not imply that each $M_i \in M$ ends up in the possession of the same key S_n.

Definition 3 (CGKA) *P provides* complete group key authentication (CGKA) *if, for every $M_i \in M$ and $M_j \in M$, with $i \neq j$, M_i and M_j compute the same key S_n only if S_n has been contributed by each $M_k \in M$ (assuming that M_i and M_j have the same view of group membership).*

CGKA implies only that an intruder cannot alter group membership by excluding some members from participation in key agreement.

Definition 4 (KC) *P provides* key confirmation (KC) *if a principal is assured that its peers have possession of a particular secret key.*

KC implies that the shared secret key, S_n, is part of the common knowledge of all the members of the group.

Definition 5 (KI) *A contributory key agreement protocol provides* key integrity *if a party is assured that its secret key is a function of* only *the individual contributions of all parties.*

Definition 6 (PFS) *P provides* perfect forward secrecy (PFS) *if compromise of a long-term key cannot result in compromise of past-session keys.*

A.2 Contract Signing Protocols

Let P be a multi-party exchange protocol and $M = \{M_1, \ldots, M_n\}$ be the set of parties. Then:

Definition 7 (Fairness) *P is* fair *if the exchange is carried out in such a way that, at the end of the protocol, every honest participant has received all the expected items corresponding to the items he has provided.*

Definition 8 (Online and offline server) *Depending on its type of involvement, a trusted third party, called* server *for short, may be* online *or* offline. *An* online server *is involved in every instance of the protocol. Conversely, an* offline TTP *is involved in some instances of the protocol, since the participants are assumed to be honest enough to do not need the TTP intervention for achieving fairness. The TTP will be involved only if problems arise. Offline TTP protocols are preferred over online TTP ones*

Definition 9 (Optimism) *P is said to be* optimistic *if it uses an offline TTP.*

Definition 10 (Timeliness) *P satisfies* timeliness *if any party can decide when to finish a protocol run without loosing fairness.*

Definition 11 (Non-repudiation) *P satisfies* non-repudiation *if no party can deny its action.*

Definition 12 (Verifiability of TTP) *P satisfies* verifiability of TTP *if the TTP misbehaves, all harmed parties will be able to prove it.*

Definition 13 (Transparency of TTP) *P satisfies* transparency of TTP *if the result of executing the exchange protocol does not depend on whether or not the TTP is involved.*

Definition 14 (Abuse-Freeness) *P satisfies* abuse-Freeness *if it is not possible for an attacker (being it either an insider or an outsider) to show a third party that the contract final state is under its control.*

Definition 15 (Completeness) *P is sais to be* complete *if the adversary cannot prevent a group of honest participants from getting a contract validly signed.*

Definition 16 (Synchronous and Asynchronous Networks and Protocols) *A network is* synchronous *if there is a limited time for a message to reach its destination (otherwise it has been lost and the transport layer manages these events) even if an attack occurs and* asynchrnous *otherwise. Synchronous (respectively asynchronous) networks use* synchronous *(respectively* asynchronous*) protocols. In the specification of an asynchronous protocol, care has to be taken to deal with message loss and unsorted message arrival. This is not the case for a synchronous protocol, in which it is further assumed that one can use a construct for testing whether a message has not been sent by other party just because it did not arrive within the limited time. For this to make sense, clocks are assumed to be synchronized.*

A.3 Attacks

Definition 17 (disruption attack) *A* disruption attack *is an attack whereby the (honest) users are prevented from completing a successful run of the protocol.*

Definition 18 (known key attack) *A* known-key attack *is an attack whereby compromise of a session key results in compromise of other session keys or impersonation of one of the (honest) parties.*

Definition 19 (unknown shared key attack) *A* unknown shared key attack *is an attack whereby if no disagreement is caused amongst parties about who is participating in the key exchange.*