A Concrete-Security Analysis of the Apple PSI Protocol

Mihir Bellare
Department of Computer Science and Engineering
University of California, San Diego

July 30, 2021

Contents

1 Introduction

The National Center for Missing and Exploited Children (NCMEC) explains that "United States federal law defines child pornography as any visual depiction of sexually explicit conduct involving a minor [2]." They refer to these images as Child Sexual Abuse Material (CSAM). They document the harm they cause and note that "the disturbing reality is that the Internet platforms we use every day ... are now being used to ... collect CSAM."

Apple is aiming to limit CSAM on its platforms. Apple users (also called clients) store photos in iCloud. Apple would like to detect if any of these photos belongs to NCMEC's database of CSAM photos. If the number of these matches exceeds some pre-determined threshold, indicating systematic presence of CSAM, Apple will report the user to appropriate authorities.

Taking action to limit CSAM is a laudable step. But its implementation needs some care. Naively done, it requires scanning the photos of all iCloud users. But our photos are personal, recording events, moments and people in our lives. Users expect and desire that these remain private from Apple. Reciprocally, the database of CSAM photos should not be made public or become known to the user.

Apple has found a way to detect and report CSAM offenders while respecting these privacy constraints. When the number of user photos that are in the CSAM database exceeds the threshold, the system is able to detect and report this. Yet a user photo that is not in the CSAM database remains invisible to the system, and users do not learn the contents of the CSAM database.

This is done using cryptography. Specifically, Apple intends to leverage a well-established cryptographic tool called Private Set Intersection (PSI).

Apple holds a set X of fingerprints of photos from a database of CSAM photos. (The CSAM database itself is held by NCMEC or other child safety organizations.) Let D be the set of fingerprints of the photos of a user. Using PSI, Apple would end up learning the intersection $I = X \cap D$ of these sets, but nothing about the photos in D whose fingerprints are not in the intersection. So the privacy of innocent user photos is protected, yet Apple can determine the number of user photos that are in the CSAM database; this is just the size of I. If this number exceeds the threshold set by policy, it can report the user. Meanwhile the user does not learn the contents of set X and hence of the CSAM database.

The Apple protocol, described in [11], adds to this some new, and innovative, elements, for two reasons. The first is to enhance privacy. The user-privacy offered by PSI is already high, but Apple goes further; its protocol denies Apple even the contents of the set I when its size is below threshold. The second reason is system and functionality constraints. Typical PSI protocols give the output to the client, but Apple needs it to go to the server; typical PSI protocols process the members of the client set (here, the fingerprints of user photos) together, but Apple needs to be able to process them individually and independently of each other.

For the protocol to provide the desired or claimed privacy, the cryptography needs to be right. How do we know that the math works, meaning that the privacy goals are met? Cryptographers assess this by giving what are called proofs of security. Such a proof establishes that the protocol meets a certain mathematical *definition* of security, assuming cryptographic primitives (building blocks) used within are themselves secure.

A security proof establishes security in an abstract model, ruling out attacks in that model. It does not cover everything that can go wrong in a real system, but it covers a lot. There is enough conviction in the community that proofs are important that, for example, they tend to be a requirement for protocols to be standardized.

Apple has provided such a proof for its protocol. Their document [11] shows that the protocol meets the canonical definition in the broad area of secure computation of which PSI is a part, namely UC security [13].

The current document complements this with another proof. It shows the protocol meets alternative definitions of security. The proof uses different methods. Most importantly, I give what cryptographers call a concrete-security analysis. This evaluates security quantitatively, giving evidence that the protocol is not only secure in principle, but is so for the key sizes in actual use.

Why another analysis? The Apple protocol will see hundreds of millions of uses. It is desirable that it receive extensive analysis, done by different people, using different methods.

In mathematics, confidence in a theorem grows with time as different people give different proofs based on different approaches. It is the same with cryptographic protocols.

Meanwhile, there is growing recognition of the importance, for practical security, of analyses that are concrete and give bounds that are tight [1, 10]. This document is a start to addressing these goals for the Apple protocol.

We start with definitions of security for the protocol that associate to an attacker a number, called its advantage, that is its probability of violating security. (There are two dominant definitional paradigms in cryptography. One, called simulation, is what underlies UC-security. The other, called indistinguishability, is what we use. Indistinguishability is more directly amenable to a concrete security treatment.)

We then give theorems that, given an adversary attacking a certain security property (as captured by one of our definitions), return a formula that upper bounds the advantage of this adversary as a sum of terms. These terms could be functions of protocol parameters, like the group size, or they could themselves be advantages, of adversaries (that the theorem proof constructs) that aim to defeat the security of cryptographic primitives used in the protocol.

The terms in the upper bound can be estimated, using cryptanalytic knowledge of the primitives and the values of protocol parameters. Once this is done, the formula yields a numerical upper bound on the advantage of our original adversary. The bounds in our results are good enough that we see evidence of security for the Apple protocol for the key and group sizes in use in the implementation.

The protocol we are analyzing here, that we call the basic Apple protocol, is the tPSI-AD protocol of [11, Section 4.1], with some simplifications noted in Section 4. We do not consider the ftPSI-AD protocol of [11, Section 4.2], but it should only offer security guarantees greater than those of the basic Apple protocol.

Now let us overview the more technical sections that follow. In Section 3 we give a protocol syntax to capture the different functional requirements that the protocol targets. This allows us to describe the basic Apple protocol in a modular way. Our description, in Figure 1, is quite compact.

In Section 5 we give indistinguishability-based definitions of security that associate to adversaries their advantage in winning a certain game that is described in pseudocode. This is for both server security (asking that clients not learn the content of the set X) and client security (asking that the server, Apple, not learn anything about client photos whose fingerprints are not in X, and, when the number of these is below threshold, not even which ones they are).

For both server and client security, we then give theorems that state explicit bounds on adversary advantage, as described above. From these, again as stated above, one can obtain numerical bounds on adversary advantage as a function of adversary running time and the key or group sizes used in the protocol implementation.

Correctness refers to a protocol performing its task, meaning returning the desired outputs given any inputs. It is usually easy to see, but, for the Apple protocol, for reasons discussed further in Section 9, it is more involved. We treat it in Section 9, giving a game-based definition, and a theorem that bounds the advantage of an adversary in violating correctness.

The proofs of the theorems use the game-playing technique [8], which breaks the proof up into a sequence of games described in pseudocode. One merit of this style is that it lends itself to automated proof verification, which would be an interesting step for the future.

We use many key technical ideas from [11]. Probably the most important is the extraction strategy of Figure 7, which is taken from the simulator in the proof of [11, Theorem 4].

2 Notation and preliminaries

By ε we denote the empty string. By $|Z|$ we denote the length of a string Z. By $x\|y$ we denote the concatenation of strings x, y.

If q is an integer then $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$ and $\mathbb{Z}_q^* = \{\, x \in \mathbb{Z}_q \;:\; \gcd(x, q) = 1 \,\}$. So if q is prime then $\mathbb{Z}_q^* = \{1, 2, \ldots, q-1\}$. We let $[1..n] = \{1, 2, \ldots, n\}$.

If S is a finite set, then $|S|$ denotes it size.

If X is a finite set, we let $x \leftarrow\!\!\$\, X$ denote picking an element of X uniformly at random and assigning it to x. Algorithms may be randomized unless otherwise indicated. If A is an algorithm, we let $y \leftarrow A[\mathrm{O}_1, \ldots](x_1, \ldots; \omega)$ denote running A on inputs $x_1, \ldots$ and coins ω, with oracle access to $\mathrm{O}_1, \ldots$, and assigning the output to y. By $y \leftarrow\!\!\$\, A[\mathrm{O}_1, \ldots](x_1, \ldots)$ we denote picking ω at random and letting $y \leftarrow A[\mathrm{O}_1, \ldots](x_1, \ldots; \omega)$. An adversary is an algorithm. Running time is worst case, which for an algorithm with access to oracles means across all possible replies from the oracles. We use $\perp$ (bot) as a special symbol to denote rejection, and it is assumed to not be in $\{0, 1\}^*$.

We use the code-based game-playing framework of [8]. A game G (see Figure 2 for an example) starts with an optional INIT procedure, followed by a non-negative number of additional procedures called oracles, and ends with a FIN procedure. Execution of adversary $\mathcal{A}$ with game G proceeds as follows. First, INIT executes, and its output is the input for $\mathcal{A}$. Now the latter executes, making calls to the game oracles. Eventually, it halts with some output. The adversary output is the input to FIN, and the output of FIN is the game output. By $\Pr[\mathrm{G}(\mathcal{A})]$ we denote the probability that the execution of game G with adversary $\mathcal{A}$ results in this game output being the boolean true.

Different games may have procedures (oracles) with the same names. If we need to disambiguate, we may write G.O to refer to oracle O of game G.

We adopt the convention that the running time of an adversary refers to the time for the execution of the game with the adversary. So the running time of the adversary includes the time taken by game procedures.

In games, integer variables, set variables boolean variables and string variables are assumed initialized, respectively, to 0, the empty set $\emptyset$, the boolean false and $\bot$.

3 Protocol syntax

The syntax allow us to break the protocol into components so as to more formally capture some of the requirements and constraints listed in [11, Section 2.3]. Breaking up the protocol into components in this way also allows a modular description of the protocol as shown in Figure 1.

The syntax captures a process in which the server (using algorithm SePost) first posts some data, and the client (using algorithm ClInit) then initializes some information that it will keep static while the protocol executes. After that, at any time, the client (using algorithm ClVch) can create and upload a voucher. This is processed (using algorithm $\mathsf{SeCollect}$) by the server. Finally, after all vouchers have been uploaded, the server (using algorithm SeOut) checks whether the number of matches is above threshold, obtaining, if so, the associated data of these matches.

Proceeding to the definition, a protocol Π specifies the following:

- U— the universe, of which the target set X will be a subset.

- HS— a set of functions, all with the same range denoted $\mathsf{HS.Rng}$. This is the space from which a random oracle $\mathrm{H} \leftarrow_{\$} \mathsf{HS}$ will be drawn. This allows the scheme to say what are the domains and ranges of the random oracles it needs. Scheme algorithms get H as oracle, indicated as an input in square brackets below.

- $(pdata, skey) \leftarrow_{\$} \mathsf{SePost}[\mathrm{H}](X)$— The server posting algorithm takes a set $X \subseteq \mathsf{U}$ to return $pdata$, that is sent to the client, and private information (a secret key) that it retains.

- $ckey \leftarrow_{\$} \mathsf{ClInit}[\mathrm{H}](pdata)$— The client initializes some client information that it will keep static during an epoch.

- $\mathit{vouch} \leftarrow\!\!\$\ \mathsf{ClVch}[\mathrm{H}](\mathit{pdata}, \mathit{ckey}, (y, \mathit{id}, \mathit{ad}))$— Given a triple consisting of a data point $y \in \mathsf{U}$, an identifier id and associated data ad, the client computes a voucher vouch that is sent to the server.
- $\mathit{sList} \leftarrow \mathsf{SeCollect}[\mathrm{H}](\mathit{pdata}, \mathit{skey}, \mathit{vouch}, \mathit{sList})$— Upon receiving a voucher vouch from the client, the server updates its list sList.
- $\mathit{sout} \leftarrow \mathsf{SeOut}[\mathrm{H}](\mathit{pdata}, \mathit{skey}, \mathit{sList})$— Once the list is complete, the server computes, from it, its final output.

We write $\Pi.\mathsf{U}$, $\Pi.\mathsf{HS}$, $\Pi.\mathsf{SePost}$, $\Pi.\mathsf{ClInit}$, $\Pi.\mathsf{ClVch}$, $\Pi.\mathsf{SeCollect}$, $\Pi.\mathsf{SeOut}$ to recover the above components from Π.

This syntax captures that the client creates vouchers one by one, carrying no state from one to the next beyond the static state ckey for the epoch. It sees the server as incrementally updating its output sList as it processes incoming vouchers.

4 The basic Apple protocol

The basic protocol, shown in Figure 1, has the following ingredients and parameters:

1. $\mathbb{G}$— a cyclic group of prime order q in which the DDH problem (as defined in Section 6) is assumed to be hard. The operation is written additively, with the identity element denoted 0.
2. G— a generator of $\mathbb{G}$
3. U— the universe, of which the target set X will be a subset.
4. $(h_1, h_2, n') \leftarrow\!\!\$\ \mathrm{MkHT}_1(|X|)$; $T \leftarrow \mathrm{MkHT}_2(X, h_1, h_2, n')$— Setup hash table for set $X \subseteq \mathsf{U}$, in two steps. The first step takes the size $n = |X|$ of X and returns an integer $n' \geq n$ together with hash functions h_1, h_2: $\mathsf{U} \to [1..n']$. The second step takes X and the outputs of the first step, and returns a function T: $[1..n'] \to X \cup \{\perp\}$ representing a table in which X is stored. The requirements are as follows. (1) For each $x \in X$ there is a unique $\ell \in \{1, 2\}$ such that $T[h_\ell(x)] = x$ and $T[h_{3-\ell}(x)] = \perp$. (2) All other table entries are $\perp$, meaning $T[i] = \perp$ if there is no $x \in X$ such that $i \in \{h_1(x), h_2(x)\}$. (3) $h_1(x) \neq h_2(x)$ for all $x \in \mathsf{U}$. The creation of the hash table is explicitly split into two steps because it is important that h_1, h_2, n' may depend on $|X|$ but not on X beyond that, and hence reveal nothing about X beyond $|X|$. This entire setup is accomplished using Cuckoo Hashing. We do not detail this, relying instead on the stated properties. We neglect the small failure probabilities associated to Cuckoo hashing.
5. A hash function H: $\mathsf{U} \to \mathbb{G} \setminus \{0\}$ that will be modeled as a random oracle.
6. t— the threshold, an integer. When the number of matches exceeds this, the server should recover their associated data.

```
$\underline{\Pi.\mathsf{SePost}[\mathrm{H}](X)\text{:}}$
$(h_1, h_2, n') \leftarrow\!\!\$\ \mathrm{MkHT}_1(|X|)$ ; $T \leftarrow \mathrm{MkHT}_2(X, h_1, h_2, n')$   // Make the hash table containing $X$
$\alpha \leftarrow\!\!\$\ [1..q-1]$ ; $L \leftarrow \alpha\, G$
For $i = 1, \ldots, n'$ do
    If $T[i] \neq \bot$ then $P_i \leftarrow \alpha\, \mathrm{H}(T[i])$ else $P_i \leftarrow\!\!\$\ \mathbb{G} \setminus \{0\}$
$pdata \leftarrow (L, n', P_1, \ldots, P_{n'}, h_1, h_2)$ ; Return $(pdata, \alpha)$

$\underline{\mathsf{ClInit}(pdata)\text{:}}$
Require: $\mathrm{Valid}(pdata) = \mathsf{true}$   // "Require" means reject if this fails. Function Valid is defined in Section 4.
$adkey \leftarrow\!\!\$\ \{0,1\}^k$ ; $a_1, \ldots, a_t \leftarrow\!\!\$\ \mathbb{F}$ ; $ckey \leftarrow (adkey, a_1, \ldots, a_t)$ ; Return $ckey$

$\underline{\mathsf{ClVch}[\mathrm{H}](pdata, ckey, (y, id, ad))\text{:}}$
$(L, n', P_1, \ldots, P_{n'}, h_1, h_2) \leftarrow pdata$   // Parse $pdata$ to recover its components
$(adkey, a_1, \ldots, a_t) \leftarrow ckey$   // Parse $ckey$ to recover its components
$adct \leftarrow\!\!\$\ \mathsf{SE.Enc}(adkey, ad)$ ; $x \leftarrow\!\!\$\ \mathbb{F} \setminus \{0\}$ ; $z \leftarrow \mathbf{P}_{adkey, a_1, \ldots, a_t}(x)$ ; $sh \leftarrow (x, z)$
$R \leftarrow \mathrm{H}(y)$ ; $w_1 \leftarrow h_1(y)$ ; $w_2 \leftarrow h_2(y)$ ; $\beta_1, \beta_2, \gamma_1, \gamma_2 \leftarrow\!\!\$\ \mathbb{Z}_q$ ; $b \leftarrow\!\!\$\ \{1, 2\}$
$Q_1 \leftarrow \beta_1\, R + \gamma_1\, G$ ; $S_1 \leftarrow \beta_1\, P_{w_b} + \gamma_1\, L$ ; $Q_2 \leftarrow \beta_2\, R + \gamma_2\, G$ ; $S_2 \leftarrow \beta_2\, P_{w_{3-b}} + \gamma_2\, L$
$K_1 \leftarrow \mathsf{KDF}(S_1)$ ; $K_2 \leftarrow \mathsf{KDF}(S_2)$ ; $ct_1 \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_1, (adct, sh))$ ; $ct_2 \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_2, (adct, sh))$
$vouch \leftarrow (id, Q_1, ct_1, Q_2, ct_2)$ ; Return $vouch$

$\underline{\mathsf{SeCollect}[\mathrm{H}](pdata, \alpha, vouch, (sList_1, sList_2))\text{:}}$
$(L, n', P_1, \ldots, P_{n'}, h_1, h_2) \leftarrow pdata$   // Parse $pdata$ to recover its components
$(id, Q_1, ct_1, Q_2, ct_2) \leftarrow vouch$   // Parse $vouch$ to recover its components
$\hat{S}_1 \leftarrow \alpha\, Q_1$ ; $\hat{S}_2 \leftarrow \alpha\, Q_2$
$K_1 \leftarrow \mathsf{KDF}(\hat{S}_1)$ ; $K_2 \leftarrow \mathsf{KDF}(\hat{S}_2)$ ; $M_1 \leftarrow \mathsf{SE.Dec}(K_1, ct_1)$ ; $M_2 \leftarrow \mathsf{SE.Dec}(K_2, ct_2)$ ; $i \leftarrow \bot$ ; $\mathrm{match} \leftarrow 0$
If $(M_1 \neq \bot$ and $M_2 = \bot)$ then $(i, \mathrm{match}) \leftarrow (1, 1)$ ; If $(M_1 = \bot$ and $M_2 \neq \bot)$ then $(i, \mathrm{match}) \leftarrow (2, 1)$
$idList \leftarrow \mathrm{Append}(idList, id)$ ; $mList \leftarrow \mathrm{Append}(mList, \mathrm{match})$
If $\mathrm{match} = 1$ then $(adct, sh) \leftarrow M_i$ ; $adList \leftarrow \mathrm{Append}(adList, adct)$ ; $shList \leftarrow \mathrm{Append}(shList, sh)$
Else $adList \leftarrow \mathrm{Append}(adList, \bot)$
Return $(idList, mList, adList, shList)$

$\underline{\mathsf{SeOut}[\mathrm{H}](pdata, \alpha, (idList, mList, adList, shList))\text{:}}$
$s \leftarrow |shList|$ ; $m \leftarrow |idList|$
If $(s < t + 1)$ then return $(idList, mList)$   // Number of matches is below threshold
$adkey \leftarrow \mathrm{Recover}(shList)$   // Shamir secret-sharing recovery to get $adkey$ from shares
For $i = 1, \ldots, m$ do
    If $(adList[i] \neq \bot)$ then $ad_i \leftarrow \mathsf{SE.Dec}(adkey, adList[i])$ else $ad_i \leftarrow \bot$
Return $idList, (ad_1, \ldots, ad_m)$
```

Figure 1: Basic Apple Protocol.

7. k— length of keys for the symmetric encryption SE, and also output length of KDF, as discussed below.

8. $\mathbb{F}$— a finite field for a Shamir secret-sharing scheme. It is required that the size $|\mathbb{F}|$ of the field be $\geq 2^k$ so that a key $K \in \{0,1\}^k$ can be represented as field element and be a secret in Shamir secret sharing.

9. $\mathbf{P}_{a_0,\ldots,a_t}$: $\mathbb{F} \to \mathbb{F}$— for $a_0, \ldots, a_t \in \mathbb{F}$, this is the polynomial defined by $\mathbf{P}_{a_0,\ldots,a_t}(x) = a_0 + a_1x + \cdots a_tx^t$ for all $x \in \mathbb{F}$. A share is a pair $(x, z) \in \mathbb{F}^2$, and this share is correct for $\mathbf{P}_{a_0,\ldots,a_t}$ if $z = \mathbf{P}_{a_0,\ldots,a_t}(x)$.

10. $s \leftarrow \text{Recover}(sh_1, \ldots, sh_n)$— Recovery of a Shamir-shared secret from a list of shares. If the shares have distinct first components and are all correct shares for $\mathbf{P}_{a_0,\ldots,a_t}$, and if $n \geq t+1$, then the output s equals a_0.

11. SE— A symmetric authenticated encryption scheme. It has key space $\{0,1\}^k$. It specifies a randomized encryption algorithm allowing encryption as $C \leftarrow\!\!\$\ \mathsf{SE.Enc}(K, M)$, and a deterministic decryption algorithm allowing decryption as $M \leftarrow \mathsf{SE.Dec}(K, C)$, and is assumed to have perfect correctness. The privacy property we need is formalized as ind\$ in Section 6. A robustness property, required for correctness, is formalized as rob\$ in Section 9. SE will be based on GCM, as described further in Section 6.

12. $sList \leftarrow \text{Append}(sList, I)$— Append item I to list $sList$.

13. KDF: $\mathbb{G} \to \{0,1\}^k$— key derivation function that derives a k-bit key $K \leftarrow \mathsf{KDF}(S)$ from a group element $S \in \mathbb{G}$. The security requirement for KDF, formalized in Section 6, asks that if the input S is random then the output $\mathsf{KDF}(S)$ is indistinguishable from random.

We now specify a protocol, in the syntax of Section 3, that we refer to as the basic Apple protocol. It is the protocol for tPSI-AD in [11, Section 4.1], with some simplifications discussed below. We do not consider the protocol of [11, Section 4.2], but it is an enhancement that adds further security, so the expectation is that it has the properties we show for the basic protocol, and possibly more.

Figure 1 shows the algorithms $\mathsf{SePost}, \mathsf{ClInit}, \mathsf{ClVch}, \mathsf{SeCollect}, \mathsf{SeOut}$. The universe U is that from the above list of ingredients. The set of functions HS is the set of all functions H: $\mathsf{U} \to \mathbb{G} \setminus \{0\}$, so that, in security games, H will be chosen randomly from this set. The function Valid invoked at line 6 is defined as follows:

```
Function Valid($pdata$)
1. $(L, P_1, \ldots, P_{n'}, h_1, h_2) \leftarrow pdata$   // Parse $pdata$ to recover its components
2. Require: $L \in \mathbb{G} \setminus \{0\}$
3. Require: $P_1, \ldots, P_{n'} \in \mathbb{G} \setminus \{0\}$ are all distinct
4. Require: $h_1, h_2$: $\mathsf{U} \to [1..n']$ and $h_1(x) \neq h_2(x)$ for all $x \in \mathsf{U}$
5. Return true if all checks pass, else false
```

The construction described in [11] enforces 4 with a syntactic condition that is easily checked, so that this check does not require evaluating the functions on all inputs in U.

This description has simplified the tPSI-AD protocol of [11, Section 4.1] in a few ways. The protocol in [11] uses a PRF applied to the identifier to obtain the x we pick at random at line 10. The purpose of the PRF is to ensure that if an identifier is re-used, then the x value is the same. We will instead assume identifiers don't repeat. Also, in the protocol in [11], the client uses an additional layer of encryption in ClVch. It is not security-relevant; its purpose is to shorten the size of the voucher. Accordingly, we omit it. As a result, our vouchers at line 14 are 4-tuples while those in [11] are 5-tuples.

5 The target security goals

Here we give definitions for the security goals that we will later show the protocol meets.

PSI is a particular case of the (two-party) secure computation problem. In its abstract and general form, party 1 has an input x_1 and party 2 has an input x_2. They want to interact and end with party 1 having output $f_1(x_1, x_2)$ and party 2 having output $f_2(x_1, x_2)$, where f_1, f_2 are fixed, public functions. Security for party 1 asks that party 2 (now an adversary) learns no more about x_1 than necessary; that is, no more than implied by the knowledge of x_2 and $f_2(x_1, x_2)$. Likewise for security for party 2.

One common way to formalize this is the UC-framework [13]. This is the approach used in [11]. It is based on the simulation paradigm.

Here we consider an alternative definition that we feel is simpler and more direct, and lends itself more easily to a concrete security treatment under which one obtains precise bounds on adversary advantage. It is based on the indistinguishability paradigm. Consider a pair of inputs $x_{1,0}, x_{1,1}$ for party 1 that satisfy $f_2(x_{1,0}, x_2) = f_2(x_{1,1}, x_2)$. Consider the protocol executed with the parties' inputs being $x_{1,c}, x_2$, where c is a random challenge bit. The output of party 2 does not depend on c due to the way $x_{1,0}, x_{1,1}$ were chosen. Security for party 1 now asks that party 2, as the adversary, not be able to determine c.

This is easy to formalize in the setting of semi-honest parties, but Apple desires security even when parties are malicious, and the protocol is designed to achieve it. In that case, what is the input x_2 of the adversary? We ask that it be determined by an extractor.

The definitions and results here are in the random oracle model [7]. The extractor needs to program the random oracle. This leads to a definitional challenge; how does one, in the absence of the distinguisher present in the simulation-based paradigm, force the extractor to program the random oracle with random values? Our answer is to require a particular form of programming in which the extractor supplies a bijection with range that of the random oracle, and the result of a query is the result of this bijection on a random input, the latter picked by the game but known to the extractor.

We now separately describe definitions for server and client security, the extraction issue arising

```
Game $\mathbf{G}^{\mathrm{ss}}_{\Pi}$

INIT:
1 $c \leftarrow\$ \{0,1\}$   // Random challenge bit
2 $\mathrm{H} \leftarrow\$ \Pi.\mathsf{HS}$   // Pick a function to be the random oracle

POST$(X_0, X_1)$:  // Adversary calls with sets $X_0, X_1$.
3 Require: $|X_0| = |X_1|$ and $X_0, X_1 \subseteq \Pi.\mathsf{U}$
4 $(pdata, skey) \leftarrow\$ \Pi.\mathsf{SePost}^{\mathrm{HASH}}(X_c)$
5 Return $pdata$   // Adversary is given this

HASH$(W)$:  // The random oracle
6 Return $\mathrm{H}(W)$

FIN$(c')$:  // Adversary provides guess $c' \in \{0,1\}$ for $c$
7 Return $(c = c')$   // Result of game, true if $c' = c$ and false otherwise
```

Figure 2: Game defining server-security of protocol Π.

only in the latter.

5.1 Server security

Server security asks that even a malicious client does not learn anything about the set X beyond its size. We formulate the requirement as an indistinguishability game, seeing X as a "message" that is being "encrypted" with the "ciphertext" being *pdata*.

The formalization considers game $\mathbf{G}^{\mathrm{ss}}_{\Pi}$ of Figure 2. The Initialize procedure picks a random challenge bit c at line 1. It also picks at random a function H to play the role of the random oracle; the random oracle itself is modeled by oracle HASH, which, given an input, simply returns the result of H on that input. The adversary $\mathcal{A}_{\mathrm{ss}}$, representing a malicious client, calls oracle POST with a pair of sets X_0, X_1. The condition at line 3 mandates that the sets must be of the same size, and be subsets of the universe, else the game rejects. At line 4, the server posting algorithm is executed on input X_c, the set designated by the challenge bit. The *pdata* generated here is returned to the adversary. We can allow the adversary multiple queries to POST to model security being maintained across multiple epochs, but for simplicity we restrict attention to one query (one epoch) for now. The adversary also has access to the random oracle HASH. Finally the adversary calls FIN with a guess c' as to the value of c. The game returns true if the guess is correct and false otherwise. The advantage of adversary $\mathcal{A}_{\mathrm{ss}}$ is:

$$\mathbf{Adv}^{\mathrm{ss}}_{\Pi}(\mathcal{A}_{\mathrm{ss}}) = 2\Pr[\mathbf{G}^{\mathrm{ss}}_{\Pi}(\mathcal{A}_{\mathrm{ss}})] - 1 \ . \quad (1)$$

A proof of security aims to bound this advantage as a function of adversary resources. The latter includes its running time and the number of queries it makes to the random oracle HASH.

5.2 Client security

This definition asks that even a malicious server does not learn anything about the client input beyond what the protocol is supposed to provide.

The definition is based on game $\mathbf{G}^{\mathrm{cs}}_{\Pi,\mathsf{Ex},\mathrm{Out}}$ of Figure 3. It is parameterized by an extractor Ex and a function Out that represents the protocol output that the server is supposed to learn. The definition is asking that it learn nothing meaningful beyond this. The advantage of an adversary $\mathcal{A}_{\mathrm{cs}}$ is:

$$\mathbf{Adv}^{\mathrm{cs}}_{\Pi,\mathsf{Ex},\mathrm{Out}}(\mathcal{A}_{\mathrm{cs}}) = 2\Pr[\mathbf{G}^{\mathrm{cs}}_{\Pi,\mathsf{Ex},\mathrm{Out}}(\mathcal{A}_{\mathrm{cs}})] - 1\ . \tag{2}$$

Client security for Π relative to Out asks that there exists an efficient extractor Ex such that $\mathbf{Adv}^{\mathrm{cs}}_{\Pi,\mathsf{Ex},\mathrm{Out}}(\mathcal{A}_{\mathrm{cs}})$ is small for all efficient $\mathcal{A}_{\mathrm{cs}}$. This is what Theorem 8.1 establishes. Let is now explain the game.

The extractor Ex specifies two algorithms, the set extractor $\mathsf{Ex.X}$ and the algorithm $\mathsf{Ex.H}$ that programs the random oracle HASH. These algorithms share a common state St that is maintained by the game. When the adversary $\mathcal{A}_{\mathrm{cs}}$ makes a query y to HASH, the game asks $\mathsf{Ex.H}$ to determine the reply. $\mathsf{Ex.H}$ does not directly provide this reply, for if it did, the game would not have a way to guarantee that it is random. Rather, $\mathsf{Ex.H}$ supplies a function f: Dom $\rightarrow$ $\mathsf{HS.Rng}$, where $\mathsf{HS.Rng}$ is the range of all functions in HS, meaning the range of the random oracle. The game picks $\omega \leftarrow\!\$\ \mathrm{Dom}$ and returns $\mathrm{f}(\omega)$ as the reply to the query. This is guaranteed to be random because f is required to be a bijection. The value ω, together with other quantities, are stored as $\mathrm{ht}[y]$, and table ht is available (read-only) to the extractor, so that it can use ω in extracting the set.

After the adversary $\mathcal{A}_{\mathrm{cs}}$ has made some number of HASH queries that are answered in this way, it will query PDATA with some *pdata*. It must make exactly one query to PDATA, and the *pdata* it submits in this query is required to satisfy the validity predicate defined above, else the game rejects at line 4, meaning the adversary automatically loses. Otherwise, the set-extraction component $\mathsf{Ex.X}$ of the extractor is run at line 5. It takes *pdata*, its current state St and the table ht. It returns a set X that it sees as underlying *pdata*, and an updated state. The extracted set X is then returned back to the adversary.

It is not the job of the definition to describe any particular extraction strategy, but, if the reader is curious, the extractor of [11], used for Theorem 8.1, is shown in our notation in Figure 7.

After its PDATA query, the adversary $\mathcal{A}_{\mathrm{cs}}$ may make more HASH queries, which continue to be answered as shown. Eventually, it will make a VOUCH query. It must make exactly one query to VOUCH. This takes the form of a pair $\mathbf{t}_0, \mathbf{t}_1$ of vectors, each a vector of triples representing a sequence of client triples. At line 10, the game takes the vector $\mathbf{t}_c$ indicated by the challenge bit c, and processes it with the protocol algorithm ClVch to get a vector $\mathbf{v}$ of client vouchers that is returned to the adversary. Perhaps after further HASH queries, the adversary returns its guess $c' \in \{0,1\}$ to the value of c, and at line 16 the game returns true iff this guess is correct.

Client vouchers do reveal some information about client triples; namely, whatever it is the purpose

```
Game $\mathbf{G}^{\mathrm{cs}}_{\Pi,\mathsf{Ex},\mathrm{Out}}$

INIT:
  $c \leftarrow\$ \{0,1\}$  // Random challenge bit
  $\mathrm{St} \leftarrow \varepsilon$  // Extractor state, explicitly maintained by game
  $\mathrm{HT}[\cdot], \mathrm{ht}[\cdot] \leftarrow \bot$  // Initialize all entries of these tables to $\bot$

PDATA($pdata$):  // Adversary calls with $pdata$
  Require: $\mathrm{Valid}(pdata) = \mathsf{true}$
  $(X, \mathrm{St}) \leftarrow\$ \mathsf{Ex.X}(pdata, \mathrm{St}, \mathrm{ht})$  // Extract set $X$ from $pdata$
  Return $X$  // Adversary gets the extracted set

VOUCH($\mathbf{t}_0, \mathbf{t}_1$):  // Adversary calls with a pair of vectors of triples
  $\mathrm{out} \leftarrow \mathrm{Out}(\mathbf{t}_0, \mathbf{t}_1, X)$
  Require: $\mathrm{out} \neq \bot$
  $ckey \leftarrow\$ \mathsf{ClInit}[\mathrm{HASH}](pdata)$
  For $i = 1, \ldots, |\mathbf{t}_0|$ do $\mathbf{v}[i] \leftarrow\$ \mathsf{ClVch}[\mathrm{HASH}](pdata, ckey, \mathbf{t}_c[i])$
  Return $\mathbf{v}, \mathrm{out}$

HASH($y$):  // The programmable random oracle
  If $\mathrm{HT}[y] \neq \bot$ then return $\mathrm{HT}[y]$
  $(\mathrm{f}, \mathrm{St}) \leftarrow\$ \mathsf{Ex.H}(y, \mathrm{St}, \mathrm{ht})$  // f: $\mathrm{f.Dom} \to \mathsf{HS.Rng}$ is a bijection
  $\omega \leftarrow\$ \mathrm{f.Dom}$ ; $\mathrm{HT}[y] \leftarrow \mathrm{f}(\omega)$ ; $\mathrm{ht}[y] \leftarrow (\mathrm{f}, \omega, \mathrm{HT}[y])$
  Return $\mathrm{HT}[y]$

FIN($c'$):  // Adversary provides guess $c' \in \{0,1\}$ for $c$
  Return $(c = c')$  // Result of game, true if $c' = c$ and false otherwise
```

```
Function $\mathrm{Out}(\mathbf{t}_0, \mathbf{t}_1, X)$
Require: $|\mathbf{t}_0| = |\mathbf{t}_1|$  // The vectors must have the same length
$m \leftarrow |\mathbf{t}_0|$  // Length of both vectors, the number of triples
Require: $\mathrm{ID}(\mathbf{t}_0) = \mathrm{ID}(\mathbf{t}_1)$  // Identifiers in the vectors are the same
Require: $\mathrm{ID}(\mathbf{t}_0[1]), \ldots, \mathrm{ID}(\mathbf{t}_0[m])$ are all distinct  // Identifiers must be distinct
Require: $\mathrm{Match}(\mathbf{t}_0, X) = \mathrm{Match}(\mathbf{t}_1, X)$  // Locations of matches in the vectors must be the same
$s \leftarrow |\mathrm{ID}(\mathbf{t}_0, X)|$  // Number of matches. By above, $s$ also equals $|\mathrm{ID}(\mathbf{t}_1, X)|$
If $s \leq t$ then  // Number of matches is below threshold
    $\mathrm{out} \leftarrow (\mathrm{ID}(\mathbf{t}_0), \mathrm{Match}(\mathbf{t}_0, X))$  // Output the locations of the matches
Else  // $s \geq t+1$, number of matches is above threshold
    Require: $\mathrm{AD}(\mathbf{t}_0, X) = \mathrm{AD}(\mathbf{t}_1, X)$  // Associated data of matches must be the same
    $\mathrm{out} \leftarrow (\mathrm{ID}(\mathbf{t}_0), \mathrm{AD}(\mathbf{t}_0, X))$  // Output the identifiers and associated data of the matches
Return out  // This is $\bot$ if any requirement above fails
```

Figure 3: **Top:** Game defining client-security of protocol Π relative to extractor Ex and output function Out. **Bottom:** The output function for the protocol we consider.

of the protocol to provide to the server. So if the adversary can submit arbitrary vectors to Vouch, it can trivially win. Accordingly, the vectors are restricted to agree on whatever information the protocol is supposed to obtain about them. This is enforced through function Out. It is called at line 7, and, at line 8, the game rejects if the result out is $\bot$. Once this restriction has been made, the definition is non-trivial, asking that nothing be revealed about client triples beyond the minimal information the protocol is allowed to get.

The particular function Out used here is shown at the bottom of Figure 3. To explain it, we need some notation. If (y, id, ad) is a client triple, then we recover its data as $\mathrm{Data}((y, id, ad)) = y$, its identifier as $\mathrm{ID}((y, id, ad)) = id$ and its associated data as $\mathrm{AD}((y, id, ad)) = ad$. This notation is extended to a m-vector $\mathbf{t}$ of triples by setting

$$\begin{aligned}\mathrm{Data}(\mathbf{t}) &= (\mathrm{Data}(\mathbf{t}[1]), \ldots, \mathrm{Data}(\mathbf{t}[m]))\\ \mathrm{ID}(\mathbf{t}) &= (\mathrm{ID}(\mathbf{t}[1]), \ldots, \mathrm{ID}(\mathbf{t}[m]))\\ \mathrm{AD}(\mathbf{t}) &= (\mathrm{AD}(\mathbf{t}[1]), \ldots, \mathrm{AD}(\mathbf{t}[m])) .\end{aligned}$$

Now for a set X we let $\mathrm{ID}(\mathbf{t}, X)$ be the set of all $\mathrm{ID}(\mathbf{t}[i])$ such that $\mathrm{Data}(\mathbf{t}[i]) \in X$. This is the set of identifiers of matches, and its size is the number of matches. We let $\mathrm{Match}((y, id, ad), X) = 1$ if $y \in X$ and 0 otherwise. We let $\mathrm{AD}((y, id, ad), X) = ad$ if $y \in X$ and $\bot$ otherwise. These also are extended to an m-vector $\mathbf{t}$ via

$$\begin{aligned}\mathrm{Match}(\mathbf{t}, X) &= (\mathrm{Match}(\mathbf{t}[1], X), \ldots, \mathrm{Match}(\mathbf{t}[m], X))\\ \mathrm{AD}(\mathbf{t}, X) &= (\mathrm{AD}(\mathbf{t}[1], X), \ldots, \mathrm{AD}(\mathbf{t}[m], X)) .\end{aligned}$$

Now we can return to function Out in Figure 3. It requires that the vectors $\mathbf{t}_0, \mathbf{t}_1$ agree in their identifiers, since these are always revealed by the protocol to the server. If the number s of matches is below threshold, the output out contains only the locations of the matches. When it is above threshold, the output contains the associated data of the matches. Throughout we assume that all associated data has the same length, and this length is public.

6 Cryptographic primitives and assumptions

Server security is based on the assumed hardness of the DDH problem in group $\mathbb{G}$. We formalize this via game $\mathbf{G}^{\mathrm{ddh}}_{\mathbb{G},G,s}$ of Figure 4. It is associated to group $\mathbb{G}$, generator G of $\mathbb{G}$ and an integer s representing the number of instances. If $\mathcal{A}_{\mathrm{ddh}}$ is an adversary playing this game, its advantage is

$$\mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},G,s}(\mathcal{A}_{\mathrm{ddh}}) = 2\Pr[\mathbf{G}^{\mathrm{ddh}}_{\mathbb{G},G,s}(\mathcal{A}_{\mathrm{ddh}})] - 1 . \quad (3)$$

It is convenient for our proofs to have made the number of instances s a parameter of the game. The standard DDH problem sets $s = 1$. A self-reducibility argument from [17], however, shows that the hardness of the s-instance version reduces tightly to the hardness of the single-instance version as long as s stays somewhat below the adversary running time, and accordingly we work

```
Game $\mathbf{G}^{\mathrm{ddh}}_{\mathbb{G},G,s}$

INIT:
1 $d \leftarrow\$ \{0,1\}$   // Random challenge bit
2 $\alpha \leftarrow\$ \mathbb{Z}_q^*$   // $q = |\mathbb{G}|$
3 $L \leftarrow \alpha G$
4 For $i = 1, \ldots, s$ do
5     $R_i \leftarrow\$ \mathbb{G} \setminus \{0\}$
6     If $d = 1$ then $S_i \leftarrow \alpha R_i$ else $S_i \leftarrow\$ \mathbb{G} \setminus \{0\}$
7 Return $(L, R_1, S_1, \ldots, R_s, S_s)$

FIN$(d')$:  // Adversary provides guess $d' \in \{0,1\}$ for $d$
8 Return $(d = d')$   // Result of game, true if $d' = d$ and false otherwise
```

Figure 4: Game defining DDH problem for group $\mathbb{G}$, generator G and number of instances $s \geq 1$.

with the former.

Game $\mathbf{G}^{\mathrm{kdf}}_{\mathsf{KDF},u}$ of Figure 5 formalizes the security assumption made on the key-derivation function $\mathsf{KDF}\colon \mathbb{G} \to \{0,1\}^k$. It asks that $\mathsf{KDF}(S)$ is indistinguishable from a random k-bit string when $S \leftarrow\$ \mathbb{G}$ is not known to the adversary. If $\mathcal{A}_{\mathrm{kdf}}$ is an adversary playing this game, its advantage is

$$\mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{KDF},u}(\mathcal{A}_{\mathrm{kdf}}) = 2\Pr[\mathbf{G}^{\mathrm{kdf}}_{\mathsf{KDF},u}(\mathcal{A}_{\mathrm{kdf}})] - 1 \ . \tag{4}$$

The definition is in the multi-user setting, with parameter u being the number of users, because this is what arises in our results and proofs. A standard hybrid argument shows that single-user ($u = 1$) security implies multi-user security with a factor u loss in advantage. However, certain choices of KDF may avoid this loss and have multi-user and single-user security that are comparable.

The symmetric encryption scheme SE draws its key K at random from key space $\{0,1\}^k$. It specifies a randomized encryption algorithm that encrypts message M to ciphertext C via $C \leftarrow\$ \mathsf{SE.Enc}(K, M)$. The length of C is $|C| = |M| + e$ where e is an integer, called the ciphertext-length overhead, that is associated to the scheme. The deterministic decryption algorithm recovers via $M \leftarrow \mathsf{SE.Dec}(K, C)$. Game $\mathbf{G}^{\mathrm{ind\$}}_{\mathsf{SE},u}$ of Figure 5 formalizes the security assumption made on SE. It asks that a ciphertext encrypting M be indistinguishable from a random string of length $|M| + e$ as long as the adversary is given neither the key nor the randomness underlying the encryption. This is called indistinguishability from random, and denoted IND\$. If $\mathcal{A}_{\mathrm{se}}$ is an adversary playing this game, its advantage is

$$\mathbf{Adv}^{\mathrm{ind\$}}_{\mathsf{SE},u}(\mathcal{A}_{\mathrm{se}}) = 2\Pr[\mathbf{G}^{\mathrm{ind\$}}_{\mathsf{SE},u}(\mathcal{A}_{\mathrm{se}})] - 1 \ . \tag{5}$$

The definition is in the multi-user setting, with parameter u being the number of users, because this is what arises in our results and proofs. A standard hybrid argument shows that single-user ($u = 1$) security implies multi-user security with a factor u loss in advantage [6]. However, certain

```
Game $\mathbf{G}^{\mathrm{kdf}}_{\mathsf{KDF},u}$

INIT:
1 $d \leftarrow\!\!\$\ \{0,1\}$   // Random challenge bit
2 For $i = 1, \ldots, u$ do
3     $S_i \leftarrow\!\!\$\ \mathbb{G}$   // Pick a random seed
4     $K_{i,0} \leftarrow\!\!\$\ \{0,1\}^k$ ; $K_{i,1} \leftarrow \mathsf{KDF}(S_i)$
5 Return $(K_{1,d}, \ldots, K_{u,d})$

FIN($d'$):  // Adversary provides guess $d' \in \{0,1\}$ for $d$
6 Return $(d = d')$   // Result of game, true if $d' = d$ and false otherwise
```

```
Game $\mathbf{G}^{\mathrm{ind\$}}_{\mathsf{SE},u}$

INIT:
1 $d \leftarrow\!\!\$\ \{0,1\}$   // Random challenge bit
2 For $i = 1, \ldots, u$ do
3     $K_i \leftarrow\!\!\$\ \{0,1\}^k$   // Pick a random key

ENC($i, M$):  // Adversary calls with user $i$ and message $M$
4 $C_1 \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_i, M)$ ; $C_0 \leftarrow\!\!\$\ \{0,1\}^{|M|+e}$
5 Return $C_d$

FIN($d'$):  // Adversary provides guess $d' \in \{0,1\}$ for $d$
6 Return $(d = d')$   // Result of game, true if $d' = d$ and false otherwise
```

Figure 5: **Top:** Game defining multi-seed security of KDF. **Bottom:** Game defining multi-user IND\$ security of SE.

choices of SE may avoid this loss and have multi-user and single-user security that are comparable, as discussed further below.

Correctness of the protocol requires that SE satisfies a form of robustness [3] that is defined in [11] and that we call random-key robustness. It is noted in [11] that random-key robustness of SE is implied by its authenticity. This leads to requiring that SE be an authenticated encryption scheme. In Section 9 we define a multi-user version of random-key robustness, and use it in Theorem 9.1 to establish correctness of the protcol.

The Apple system is building SE from the standardized GCM authenticated encryption scheme. The latter is an AEAD scheme [18], so the encryption algorithm is deterministic and, in addition to key K and message M, takes a 96-bit nonce N and associated data A to return a ciphertext, written $Y \leftarrow \mathsf{GCM.Enc}(K, M, N, A)$. Decryption works as $M \leftarrow \mathsf{GCM.Dec}(K, Y, N, A)$, and in particular assumes possession of the nonce. To turn this into a scheme SE meeting our IND\$ definition, a natural possibility is that $\mathsf{SE.Enc}(K, M)$ pick $N \leftarrow\!\!\$\ \{0,1\}^{96}$ and return $N \| Y$

where $Y \leftarrow \mathsf{GCM.Enc}(K, M, N, \varepsilon)$. (Here $\|$ denotes concatenation and ε is the empty string.) $\mathsf{SE.Dec}(K, C)$ will parse C as $N\|Y \leftarrow C$ and return $M \leftarrow \mathsf{GCM.Dec}(K, Y, N, \varepsilon)$. An attractive feature of GCM for our purposes is that its multi-user security is close to its single-user security (as opposed to off by a factor of u) [12].

Symmetric encryption is used in two places in the protocol. Theorem 8.1 indicates that in one use, only a single message is encrypted under each key, meaning encryption is what's called one-time. In this case, it would be fine to use GCM directly with a known and constant nonce. But this will not work for the second usage, where many messages are encrypted under a single key, making randomization necessary for security. To keep things simple, the Apple system is using a single, randomized scheme SE across both usages.

7 Server-security of the protocol

Theorem 7.1 below implies that if DDH is hard then the basic Apple protocol is server-secure. The statement itself is quantitative, implying that the advantage of an adversary with certain resources in violating server security is, up to a factor of two, the same as the advantage of an adversary with comparable resources in breaking DDH. This tight reduction means there is almost no loss in security relative to DDH itself. This is evidence that clients won't obtain meaningful information about the dataset X held by Apple.

Theorem 7.1 Let Π be the basic Apple protocol (Figure 1) associated to the ingredients listed in Section 4. Let $\mathcal{A}_{\mathrm{ss}}$ be an adversary, playing game $\mathbf{G}^{\mathrm{ss}}_{\Pi}$, that makes one Post query, and let n be the size of the sets in that query. Let h be the number of Hash queries of $\mathcal{A}_{\mathrm{ss}}$. Let $s = h + n$. Then we can construct an adversary $\mathcal{A}_{\mathrm{ddh}}$, playing game $\mathbf{G}^{\mathrm{ddh}}_{\mathbb{G},G,s}$, such that

$$\mathbf{Adv}^{\mathrm{ss}}_{\Pi}(\mathcal{A}_{\mathrm{ss}}) \leq 2 \cdot \mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},G,s}(\mathcal{A}_{\mathrm{ddh}}) \, .$$

Adversary $\mathcal{A}_{\mathrm{ddh}}$, shown explicitly in Figure 6, has about the same running time as $\mathcal{A}_{\mathrm{ss}}$.

Proof of Theorem 7.1: The input to adversary $\mathcal{A}_{\mathrm{ddh}}$ of Figure 6 is the tuple returned by the Init procedure of game $\mathbf{G}^{\mathrm{ddh}}_{\mathbb{G},G,s}$. It begins by picking at random a bit c to play the role of the challenge bit in game $\mathbf{G}^{\mathrm{ss}}_{\Pi}$. It then runs $\mathcal{A}_{\mathrm{ss}}$, simulating the latter's Post, Hash oracles via the shown subroutines Post*, Hash*, respectively, to get its guess bit $c' \in \{0,1\}$. The guess d' that $\mathcal{A}_{\mathrm{ddh}}$ returns is 1 if $c' = c$ and 0 otherwise. The replies that $\mathcal{A}_{\mathrm{ddh}}$ makes to Hash* queries are $R_1, \ldots, R_s$. The table HT is assumed to be initially $\perp$ everywhere, and then gets populated so that HT$[y]$ holds the value of Hash$^*(y)$. The reason the DDH parameter is $s = h + n$ is that there are h Hash* queries by $\mathcal{A}_{\mathrm{ss}}$ and then potentially n more through line 11. For the analysis we have

$$\Pr\left[d' = 1 \,|\, d = 1\right] = \Pr\left[c' = c \,|\, d = 1\right] = \frac{1}{2} \cdot \mathbf{Adv}^{\mathrm{ss}}_{\Pi}(\mathcal{A}_{\mathrm{ss}}) + \frac{1}{2}$$

$$\Pr\left[d' = 1 \,|\, d = 0\right] = \Pr\left[c' = c \,|\, d = 0\right] = \frac{1}{2} \, .$$

```
Adversary $\mathcal{A}_{\mathrm{ddh}}(L, R_1, S_1, \ldots, R_s, S_s)$:   // Get the input from $\mathbf{G}^{\mathrm{ddh}}_{\mathbb{G},G,s}$.Init
c ←$ {0,1}   // Pick challenge bit for game $\mathbf{G}^{\mathrm{ss}}_{\Pi}$
j ← 0   // Initialize hash-query counter
$c' \leftarrow\$ \mathcal{A}_{\mathrm{ss}}[\textsc{Post}^*, \textsc{Hash}^*]$   // Run $\mathcal{A}_{\mathrm{ss}}$, simulating its oracles, to get its guess $c'$
If $(c' = c)$ then $d' \leftarrow 1$ else return $d' \leftarrow 0$
Return $d'$

Subroutine $\textsc{Post}^*(X_0, X_1)$:   // Subroutine simulating oracle $\mathbf{G}^{\mathrm{ss}}_{\Pi}$.Post
$n \leftarrow |X_0|$   // Assume $|X_0| = |X_1|$ and $X_0, X_1 \subseteq \Pi.\mathsf{U}$
$(h_1, h_2, n') \leftarrow\$ \mathrm{MkHT}_1(n)$   // Make hash table parameters
$T \leftarrow \mathrm{MkHT}_2(X_b, h_1, h_2, n')$   // Put challenge set in hash table
For $i = 1, \ldots, n'$ do
    If $T[i] \neq \bot$ then
        $R \leftarrow \textsc{Hash}^*(T[i])$   // Ensure that $T[i]$ has been queried to $\textsc{Hash}^*$ so that $\beta(T[i])$ is defined
        $P_i \leftarrow S_{\beta(T[i])}$
    Else $P_i \leftarrow\$ \mathbb{G} \setminus \{0\}$
$pdata \leftarrow (L, n', P_1, \ldots, P_{n'}, h_1, h_2)$
Return $pdata$

Subroutine $\textsc{Hash}^*(y)$:   // Subroutine simulating random oracle $\mathbf{G}^{\mathrm{ss}}_{\Pi}$.Hash
If $\mathrm{HT}[y] \neq \bot$ then return $\mathrm{HT}[y]$
$j \leftarrow j + 1$ ; $\mathrm{HT}[y] \leftarrow R_j$ ; $\beta[y] \leftarrow j$
Return $\mathrm{HT}[y]$
```

Figure 6: Adversary $\mathcal{A}_{\mathrm{ddh}}$ for proof of Theorem 7.1.

The justification for the second equation is as follows. When $d = 0$, the group elements $P_1, \ldots, P_{n'}$ are all random in $\mathbb{G} \setminus \{0\}$, independently of b. Also, h_1, h_2, n' depend only on n and thus not on b. (This is where we use this aspect of the hash table creation process.) So *pdata* is independent of c. Now, from the two equations above, we have

$$\mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},G,m}(\mathcal{A}_{\mathrm{ddh}}) = \Pr\left[d' = 1 \mid d = 1\right] - \Pr\left[d' = 1 \mid d = 0\right] = \frac{1}{2} \cdot \mathbf{Adv}^{\mathrm{ss}}_{\Pi}(\mathcal{A}_{\mathrm{ss}}) \ .$$

This yields the equation in the theorem. ▮

8 Client security of the protocol

Theorem 8.1 below implies that the protocol achieves the above definition of client security assuming the KDF and the symmetric encryption scheme meet the definitions of security given in Section 6. Equation (6) shows that the reduction is pretty tight. This means that we have evidence that the server cannot obtain meaningful information about client data, not just in principle, but even with the group and key sizes chosen in the implementation.

The theorem refers to a parameter μ. To explain, recall that the extractor returns a set X, and then the adversary makes a query to VOUCH that consists of a pair $(\mathbf{t}_0, \mathbf{t}_1)$ of vectors of triples. We can assume that $\mathrm{Out}(\mathbf{t}_0, \mathbf{t}_1) \neq \perp$, where Out is the function of Figure 3. As a consequence the number of matches in the two vectors is the same, meaning $|\mathrm{ID}(\mathbf{t}_0, X)| = |\mathrm{ID}(\mathbf{t}_1, X)|$. The theorem asks that this common size be bounded above by a parameter that it calls μ. The parameter shows up in the bound on the number of ENC calls of adversary $\mathcal{A}_{\mathrm{se},2}$. We expect μ to be much smaller than the number m of triples in the vectors. This, as we discuss more below, translates to better quantitative security for the protocol.

Theorem 8.1 Let Π be the basic Apple protocol (Figure 1) associated to the ingredients listed in Section 4, and in particular let $q = |\mathbb{G}|$ be the size of the group. Then there is an extractor Ex, shown explicitly in Figure 7, such that the following is true. Let $\mathcal{A}_{\mathrm{cs}}$ be an adversary, playing game $\mathbf{G}^{\mathrm{cs}}_{\Pi,\mathsf{Ex},\mathsf{Out}}$, that makes one PDATA query and achieves advantage $\epsilon_{\mathrm{cs}} = \mathbf{Adv}^{\mathrm{cs}}_{\Pi,\mathsf{Ex},\mathsf{Out}}(\mathcal{A}_{\mathrm{cs}})$. Let h be the number of HASH queries of $\mathcal{A}_{\mathrm{cs}}$ and m the length of each vector in its (one) VOUCH query. Let μ be a parameter such that $|\mathrm{ID}(\mathbf{t}_0, X)| \leq \mu$ whenever X is the set returned by the extractor and $(\mathbf{t}_0, \mathbf{t}_1)$ is the VOUCH query of $\mathcal{A}_{\mathrm{cs}}$. Then we can construct adversaries $\mathcal{A}_{\mathrm{kdf}}, \mathcal{A}_{\mathrm{se},1}, \mathcal{A}_{\mathrm{se},2}$ such that

$$\epsilon_{\mathrm{cs}} \leq 2 \cdot \epsilon_{\mathrm{kdf}} + 2 \cdot \epsilon_{\mathrm{se},1} + 2 \cdot \epsilon_{\mathrm{se},2} + \frac{4(m+h)}{q-1}\ , \tag{6}$$

where:

- $\epsilon_{\mathrm{kdf}} = \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{KDF},2m}(\mathcal{A}_{\mathrm{kdf}})$ is the advantage of $\mathcal{A}_{\mathrm{kdf}}$. It is playing game $\mathbf{G}^{\mathrm{kdf}}_{\mathsf{KDF},2m}$.
- $\epsilon_{\mathrm{se},1} = \mathbf{Adv}^{\mathrm{ind\$}}_{\mathsf{SE},2m}(\mathcal{A}_{\mathrm{se},1})$ is the advantage of $\mathcal{A}_{\mathrm{se},1}$. It is playing game $\mathbf{G}^{\mathrm{ind\$}}_{\mathsf{SE},2m}$ and makes one query, per each of its $2m$ users, to its ENC oracle.
- $\epsilon_{\mathrm{se},2} = \mathbf{Adv}^{\mathrm{ind\$}}_{\mathsf{SE},1}(\mathcal{A}_{\mathrm{se},2})$ is the advantage of $\mathcal{A}_{\mathrm{se},2}$. It is playing game $\mathbf{G}^{\mathrm{ind\$}}_{\mathsf{SE},1}$ and makes $\min(\mu, t)$ queries, all for its one user, to its ENC oracle, where t is the protocol threshold parameter.

The constructed adversaries all have about the same running time as $\mathcal{A}_{\mathrm{cs}}$. ▌

The extractor, shown in Figure 7, is taken from the simulator of [11]. The way it programs the random oracle is to reply to query x by βG for a $\beta \leftarrow\!\!\$\ \mathbb{G} \setminus \{0\}$ that it retains and uses in the extraction.

The extractor must be efficient, the more so, the better. We make no explicit claim in this regard. The way it shows up is that, in Theorem 8.1, the running times of the constructed adversaries include the time to run the extractor. (This is due to our convention that the running time of $\mathcal{A}_{\mathrm{cs}}$ is actually the time of the execution of this adversary with game $\mathbf{G}^{\mathrm{cs}}_{\Pi,\mathsf{Ex},\mathsf{Out}}$, and thus includes the time to run the extractor, as part of the execution time of procedure PDATA.) So higher extractor time means higher running time for the constructed adversaries, and thus their higher advantage, which decreases security according to Equation (6).

The parameter μ in the theorem statement is the number of matches, which we expect to be small.

(For an honest user, who has no CSAM, it is zero.) The theorem says that security of encryption under *adkey* is only required for the encryption of this small number μ of messages. (Despite this encryption being employed for all m triples and m being much larger than μ.) This number being small is important because, when there is just one user, adversary advantage in violating security of the encryption scheme grows quadratically in the number of messages encrypted. So the fact that μ is small significantly improves the quantitative security that we can show is provided by the protocol. (A naive analysis would set $\mu = m$, resulting in much poorer quantitative guarantees.)

The proof of Theorem 8.1 will use the following lemma. It is due to Naor and Reingold [17], and is discussed in [11, Section 3.1]. It considers a pair (Q, S), constructed from P, R, and describes its distribution depending on whether or not $P = \alpha\, R$. For completeness, we give the proof, which uses linear algebra.

Lemma 8.2 Let $\mathbb{G}$ be a group of prime order q, and G a generator of $\mathbb{G}$. Let $\alpha \in \mathbb{Z}_q^*$ and let $L = \alpha\, G$. Let $P, R \in \mathbb{G} \setminus \{0\}$. Define Q, S: $\mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ by $\mathsf{Q}(\beta, \gamma) = \beta\, R + \gamma\, G$ and $\mathsf{S}(\beta, \gamma) = \beta\, P + \gamma\, L$. Regard Q, S as random variables over the random choices $\beta, \gamma \leftarrow\!\!\$\, \mathbb{Z}_q$.

(1) Assume $P \neq \alpha\, R$. Then the pair (Q, S) is uniformly distributed over $\mathbb{G} \times \mathbb{G}$.

(2) Assume $P = \alpha\, R$. Then the pair (Q, S) is uniformly distributed over $\{\, (Q, S) \in \mathbb{G} \times \mathbb{G} \;:\; S = \alpha\, Q \,\}$. ▌

Proof of Lemma 8.2: Let $r \leftarrow \mathrm{DLog}_{\mathbb{G},G}(R)$ and $p \leftarrow \mathrm{DLog}_{\mathbb{G},G}(P)$. Let $t \leftarrow \mathrm{DLog}_{\mathbb{G},G}(\mathsf{Q}(\beta, \gamma))$ and $s \leftarrow \mathrm{DLog}_{\mathbb{G},G}(\mathsf{S}(\beta, \gamma))$. Then

$$\begin{bmatrix} r & 1 \\ p & \alpha \end{bmatrix} \cdot \begin{bmatrix} \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} t \\ s \end{bmatrix} .$$

The determinant of the 2 by 2 matrix on the left is $D = (r\alpha - p) \bmod q$. For part (1), the assumption that $P \neq \alpha\, R$ means that $p \neq \alpha r \bmod q$ and thus $D \neq 0$, so the matrix is invertible. So for every choice of $(t, s) \in \mathbb{Z}_q^2$ there is a unique $(\beta, \gamma) \in \mathbb{Z}_q^2$ such that the above matrix equation is true. The claim follows. For part (2), $D = 0$ and $s = (p\beta + \alpha\gamma) \bmod q = (r\alpha\beta + \alpha\gamma) \bmod q = \alpha t \bmod q$, and the claim follows. ▌

Proof of Theorem 8.1: We use a sequence of games. The Init, Pdata, Hash and Fin procedures of these games are shown in Figure 7. We give the Vouch procedure separately for each game.

In Figure 7, at line 10, α is set to the discrete log of L. Games are only used in the analysis and don't have to be efficient, making this possible. The Hash procedure replies directly according to the way the extractor programs it rather than calling the extractor.

Consider the games $\mathrm{G}_0, \mathrm{G}_1$ in Figure 8. We claim that

$$\frac{1}{2} \cdot \mathbf{Adv}^{\mathrm{cs}}_{\Pi,\mathsf{Ex},\mathsf{Out}}(\mathcal{A}_{\mathrm{cs}}) + \frac{1}{2} = \Pr[\mathrm{G}_0(\mathcal{A}_{\mathrm{cs}})] . \tag{7}$$

```
Extractor Ex.H($y$, St, ht):  // Program the random oracle
1 Define f: $\mathbb{Z}_q^* \to \mathbb{G} \setminus \{0\}$ by $\mathrm{f}(\beta) = \beta G$
2 Return (f, St)

Extractor Ex.X($pdata$, St, ht):  // Extract the set
3 Require: Valid($pdata$) = true
4 $(L, P_1, \ldots, P_{n'}, h_1, h_2) \leftarrow pdata$   // Parse $pdata$ to recover its components
5 For all $x$ such that $\mathrm{ht}[x] \neq \perp$ do
6     $(\mathrm{f}, \beta, R) \leftarrow \mathrm{ht}[x]$
7     If ( $\exists i \in \{h_1(x), h_2(x)\} : (P_i = \beta L)$ ) then $X \leftarrow X \cup \{x\}$
8 Return $(X, \mathrm{St})$
```

```
INIT:
7  $c \leftarrow\$ \{0,1\}$ ; $\mathrm{St} \leftarrow \varepsilon$ ; $\mathrm{HT}[\cdot], \mathrm{ht}[\cdot] \leftarrow \perp$
8  $adkey \leftarrow\$ \{0,1\}^k$ ; $a_1, \ldots, a_t \leftarrow\$ \mathbb{F}$ ; $ckey \leftarrow (adkey, a_1, \ldots, a_t)$

PDATA($pdata$):
9  $(X, \mathrm{St}) \leftarrow\$ \mathsf{Ex.X}(pdata, \mathrm{St}, \mathrm{ht})$
10 $(L, n', P_1, \ldots, P_{n'}, h_1, h_2) \leftarrow pdata$ ; $\alpha \leftarrow \mathrm{DLog}_{\mathbb{G},G}(L)$ ; Return $X$

HASH($y$):
11 If $\mathrm{HT}[y] \neq \perp$ then return $\mathrm{HT}[y]$
12 $\omega \leftarrow\$ \mathbb{Z}_q^*$ ; $\mathrm{HT}[y] \leftarrow \omega G$ ; $\mathrm{ht}[y] \leftarrow (\omega, \mathrm{HT}[y])$
13 Return $\mathrm{HT}[y]$

FIN($c'$):
14 Return $(c = c')$
```

Figure 7: **Top:** Extractor. **Bottom:** Some procedures for games in proofs.

Game G_0 follows game $\mathbf{G}^{\mathrm{cs}}_{\Pi,\mathsf{Ex},\mathsf{Out}}$, but additionally, at line 7, picks S'_1, S'_2 at random, and at line 9 uses them to perform the encryption. At line 10, if $y \in X$, it returns correctly what $\mathbf{G}^{\mathrm{cs}}_{\Pi,\mathsf{Ex},\mathsf{Out}}$ would return. If $y \notin X$ then, at line 13, it instead returns the ciphertexts based on S'_1, S'_2. However, if at line 12 the bad flag is set, then, because game G_0 includes the boxed code, the values are corrected before being returned. So the game is performing correctly if either $y \in X$ or bad = true. We now claim it is also performing correctly otherwise. For this we invoke Lemma 8.2. If bad = false and $y \notin X$ then we have $\alpha R \notin \{P_{w_1}, P_{w_2}\}$, so the condition in part (1) of the lemma is true. The lemma says that Q_1, S_1, Q_2, S_2 are randomly and independently distributed group elements. So replacing S_1, S_2 by S'_1, S'_2 makes no difference. This justifies Equation (7).

```
$\text{VOUCH}(\mathbf{t}_0, \mathbf{t}_1)$: // Games $\boxed{\mathrm{G}_0}$, $\mathrm{G}_1$
1 out $\leftarrow$ Out$(\mathbf{t}_0, \mathbf{t}_1, X)$
2 For $i = 1, \ldots, |\mathbf{t}_0|$ do
3     $(y, id, ad) \leftarrow \mathbf{t}_c[i]$
4     $adct \leftarrow\!\!\$\ \mathsf{SE.Enc}(adkey, ad)$ ; $x \leftarrow\!\!\$\ \mathbb{F} \setminus \{0\}$ ; $z \leftarrow \mathbf{P}_{adkey,a_1,\ldots,a_t}(x)$ ; $sh \leftarrow (x, z)$
5     $R \leftarrow \text{HASH}(y)$ ; $w_1 \leftarrow h_1(y)$ ; $w_2 \leftarrow h_2(y)$ ; $\beta_1, \beta_2, \gamma_1, \gamma_2 \leftarrow\!\!\$\ \mathbb{Z}_q$ ; $b \leftarrow\!\!\$\ \{1, 2\}$
6     $Q_1 \leftarrow \beta_1 R + \gamma_1 G$ ; $Q_2 \leftarrow \beta_2 R + \gamma_2 G$
7     $S_1 \leftarrow \beta_1 P_{w_b} + \gamma_1 L$ ; $S_2 \leftarrow \beta_2 P_{w_{3-b}} + \gamma_2 L$ ; $S_1' \leftarrow\!\!\$\ \mathbb{G}$ ; $S_2' \leftarrow\!\!\$\ \mathbb{G}$
8     $K_1 \leftarrow \mathsf{KDF}(S_1)$ ; $K_2 \leftarrow \mathsf{KDF}(S_2)$ ; $ct_1 \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_1, (adct, sh))$ ; $ct_2 \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_2, (adct, sh))$
9     $K_1' \leftarrow \mathsf{KDF}(S_1')$ ; $K_2' \leftarrow \mathsf{KDF}(S_2')$ ; $ct_1' \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_1', (adct, sh))$ ; $ct_2' \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_2', (adct, sh))$
10    If $y \in X$ then $\mathbf{v}[i] \leftarrow (id, Q_1, ct_1, Q_2, ct_2)$
11    Else
12        If $(\alpha R \in \{P_{w_1}, P_{w_2}\})$ then $\mathsf{bad} \leftarrow \mathsf{true}$ ; $\boxed{(ct_1', ct_2') \leftarrow (ct_1, ct_2)}$
13        $\mathbf{v}[i] \leftarrow (id, Q_1, ct_1', Q_2, ct_2')$
14 Return $\mathbf{v}$, out
```

```
$\text{VOUCH}(\mathbf{t}_0, \mathbf{t}_1)$: // Game $\mathrm{G}_2$
1 out $\leftarrow$ Out$(\mathbf{t}_0, \mathbf{t}_1, X)$
2 For $i = 1, \ldots, |\mathbf{t}_0|$ do
3     $(y, id, ad) \leftarrow \mathbf{t}_c[i]$
4     $adct \leftarrow\!\!\$\ \mathsf{SE.Enc}(adkey, ad)$ ; $x \leftarrow\!\!\$\ \mathbb{F} \setminus \{0\}$ ; $z \leftarrow \mathbf{P}_{adkey,a_1,\ldots,a_t}(x)$ ; $sh \leftarrow (x, z)$
5     $R \leftarrow \text{HASH}(y)$ ; $w_1 \leftarrow h_1(y)$ ; $w_2 \leftarrow h_2(y)$ ; $\beta_1, \beta_2, \gamma_1, \gamma_2 \leftarrow\!\!\$\ \mathbb{Z}_q$ ; $b \leftarrow\!\!\$\ \{1, 2\}$
6     $Q_1 \leftarrow \beta_1 R + \gamma_1 G$ ; $Q_2 \leftarrow \beta_2 R + \gamma_2 G$
7     $S_1 \leftarrow \beta_1 P_{w_b} + \gamma_1 L$ ; $S_2 \leftarrow \beta_2 P_{w_{3-b}} + \gamma_2 L$
8     $K_1 \leftarrow \mathsf{KDF}(S_1)$ ; $K_2 \leftarrow \mathsf{KDF}(S_2)$ ; $ct_1 \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_1, (adct, sh))$ ; $ct_2 \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_2, (adct, sh))$
9     $K_1' \leftarrow\!\!\$\ \{0,1\}^k$ ; $K_2' \leftarrow\!\!\$\ \{0,1\}^k$ ; $ct_1' \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_1', (adct, sh))$ ; $ct_2' \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_2', (adct, sh))$
10    If $y \in X$ then $\mathbf{v}[i] \leftarrow (id, Q_1, ct_1, Q_2, ct_2)$
11    Else $\mathbf{v}[i] \leftarrow (id, Q_1, ct_1', Q_2, ct_2')$
12 Return $\mathbf{v}$, out
```

Figure 8: Games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2$ for proof of Theorem 8.1.

Games $\mathrm{G}_0, \mathrm{G}_1$ are identical-until-bad. So by the Fundamental Lemma of Game Playing [8] we have

$$\Pr[\mathrm{G}_0(\mathcal{A}_{\mathrm{cs}})] = \Pr[\mathrm{G}_1(\mathcal{A}_{\mathrm{cs}})] + (\Pr[\mathrm{G}_0(\mathcal{A}_{\mathrm{cs}})] - \Pr[\mathrm{G}_1(\mathcal{A}_{\mathrm{cs}})]) \quad (8)$$

$$\leq \Pr[\mathrm{G}_1(\mathcal{A}_{\mathrm{cs}})] + \Pr[\mathrm{G}_1(\mathcal{A}_{\mathrm{cs}}) \text{ sets } \mathsf{bad}] . \quad (9)$$

Now we need to bound both terms in Equation (9).

Consider the computation $R \leftarrow \text{HASH}(y)$ at line 5. If $\mathrm{HT}[y]$ had been defined before $L, P_1, \ldots, P_{n'}$,

h_1, h_2 were submitted to PDATA, then line 12 will not set bad because $y \notin X$. So now consider HASH queries made by $\mathcal{A}_{\mathrm{cs}}$ after it submitted $L, P_1, \ldots, P_{n'}, h_1, h_2$. Since $\mathcal{A}_{\mathrm{cs}}$ could pick the y in $\mathbf{t}_b$ at line 3 after making these queries, and since each query has a $2/(q-1)$ chance of setting bad at line 12, the overall chance that $\mathcal{A}_{\mathrm{cs}}$ can set bad at line 12 through its h queries to HASH is at most $2h/(q-1)$. Finally, if $\mathcal{A}_{\mathrm{cs}}$ did not query HASH(y), the query at line 5 has a further $2/(q-1)$ chance of setting bad at line 12, and there are m such queries. So the overall chance of setting bad is

$$\Pr[\mathrm{G}_1(\mathcal{A}_{\mathrm{cs}}) \text{ sets } \mathsf{bad}] \leq \frac{2(m+h)}{q-1} \ . \tag{10}$$

In game G_1, the random choice of S_1', S_2' for $y \notin X$ allows us to exploit the assumed security of KDF to replace K_1', K_2' by random k-bit strings. This is the change made in moving to G_2 of Figure 8, showing up at line 9 of the latter. We construct adversary $\mathcal{A}_{\mathrm{kdf}}$, playing game $\mathbf{G}^{\mathrm{kdf}}_{\mathsf{KDF},2m}$, so that

$$\Pr[\mathrm{G}_1(\mathcal{A}_{\mathrm{cs}})] - \Pr[\mathrm{G}_2(\mathcal{A}_{\mathrm{cs}})] \leq \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{KDF},2m}(\mathcal{A}_{\mathrm{kdf}}) \ . \tag{11}$$

Adversary $\mathcal{A}_{\mathrm{kdf}}$ receives, from $\mathbf{G}^{\mathrm{kdf}}_{\mathsf{KDF},2m}$.INIT, a list of $2m$ k-bit keys that we write as $K_{1,1}, K_{1,2}, \ldots, K_{m,1}, K_{m,2}$. Adversary $\mathcal{A}_{\mathrm{kdf}}$ starts by executing the steps of INIT in Figure 7. It then runs $\mathcal{A}_{\mathrm{cs}}$, itself replying to the latter's oracle queries, to get $\mathcal{A}_{\mathrm{cs}}$'s output bit c', and returns the latter as its own guess bit. It replies to PDATA and HASH queries of $\mathcal{A}_{\mathrm{cs}}$ as per Figure 7. For the VOUCH query, it proceeds as in game G_2, executing lines 1–8. At line 9, instead of picking K_1', K_2', it sets $K_1' \leftarrow K_{i,1}$ and $K_2' \leftarrow K_{i,2}$. (Here i is the For loop counter from line 2.) It then reverts to following game G_2 for lines 10–12, the response to the query being as per line 12. For the analysis, let $S_{1,1}, S_{1,2}, \ldots, S_{m,1}, S_{m,2}$ denote the seeds chosen at line 3 of $\mathbf{G}^{\mathrm{kdf}}_{\mathsf{KDF},2m}$.INIT. Then $S_{i,1}, S_{i,2}$ play the role of S_1', S_2' in game G_1. So if the challenge bit d of game $\mathbf{G}^{\mathrm{kdf}}_{\mathsf{KDF},2m}$ is 1 then $\mathcal{A}_{\mathrm{cs}}$ gets the view it would in game G_1, and if d is 0 then $\mathcal{A}_{\mathrm{cs}}$ gets the view it would in game G_2, which implies Equation (11).

Now, using Equation (11), we have

$$\begin{aligned}
\Pr[\mathrm{G}_1(\mathcal{A}_{\mathrm{cs}})] &= \Pr[\mathrm{G}_2(\mathcal{A}_{\mathrm{cs}})] + (\Pr[\mathrm{G}_1(\mathcal{A}_{\mathrm{cs}})] - \Pr[\mathrm{G}_2(\mathcal{A}_{\mathrm{cs}})]) \qquad (12)\\
&\leq \Pr[\mathrm{G}_2(\mathcal{A}_{\mathrm{cs}})] + \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{KDF},2m}(\mathcal{A}_{\mathrm{kdf}}) \ , \qquad (13)
\end{aligned}$$

so our task is now to bound $\Pr[\mathrm{G}_2(\mathcal{A}_{\mathrm{cs}})]$.

In game G_2, the random choice of K_1', K_2' for $y \notin X$ allows us to exploit the assumed IND\$-security of SE to replace ct_1', ct_2' by random strings. This is the change made in moving to G_3 of Figure 9, showing up at line 9 of the latter. Here ℓ denotes the length of the message $(adct, sh)$ that is being encrypted, and e is the ciphertext-length overhead of SE. (We are using the assumption that the length of the associated data is a fixed, known constant across all queries, which means that the length of the messages being encrypted is also the same across all queries.) We construct adversary

```
VOUCH($\mathbf{t}_0, \mathbf{t}_1$):  // Game $G_3$
out $\leftarrow$ Out($\mathbf{t}_0, \mathbf{t}_1, X$)
For $i = 1, \ldots, |\mathbf{t}_0|$ do
    $(y, id, ad) \leftarrow \mathbf{t}_c[i]$
    $adct \leftarrow\!\!\$\ \mathsf{SE.Enc}(adkey, ad)$ ; $x \leftarrow\!\!\$\ \mathbb{F} \setminus \{0\}$ ; $z \leftarrow \mathbf{P}_{adkey, a_1, \ldots, a_t}(x)$ ; $sh \leftarrow (x, z)$
    $R \leftarrow \textsc{Hash}(y)$ ; $w_1 \leftarrow h_1(y)$ ; $w_2 \leftarrow h_2(y)$ ; $\beta_1, \beta_2, \gamma_1, \gamma_2 \leftarrow\!\!\$\ \mathbb{Z}_q$ ; $b \leftarrow\!\!\$\ \{1, 2\}$
    $Q_1 \leftarrow \beta_1 R + \gamma_1 G$ ; $Q_2 \leftarrow \beta_2 R + \gamma_2 G$
    $S_1 \leftarrow \beta_1 P_{w_b} + \gamma_1 L$ ; $S_2 \leftarrow \beta_2 P_{w_{3-b}} + \gamma_2 L$
    $K_1 \leftarrow \mathsf{KDF}(S_1)$ ; $K_2 \leftarrow \mathsf{KDF}(S_2)$ ; $ct_1 \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_1, (adct, sh))$ ; $ct_2 \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_2, (adct, sh))$
    $ct_1' \leftarrow\!\!\$\ \{0,1\}^{\ell+e}$ ; $ct_2' \leftarrow\!\!\$\ \{0,1\}^{\ell+e}$
    If $y \in X$ then $\mathbf{v}[i] \leftarrow (id, Q_1, ct_1, Q_2, ct_2)$
    Else $\mathbf{v}[i] \leftarrow (id, Q_1, ct_1', Q_2, ct_2')$
Return $\mathbf{v}$, out
```

Figure 9: Game G_3 for proof of Theorem 8.1.

$\mathcal{A}_{\rm se,1}$, playing game $\mathbf{G}^{\rm ind\$}_{\mathsf{SE},2m}$, so that

$$\Pr[G_2(\mathcal{A}_{\rm cs})] - \Pr[G_3(\mathcal{A}_{\rm cs})] \le \mathbf{Adv}^{\rm ind\$}_{\mathsf{SE},2m}(\mathcal{A}_{\rm se,1}) \ . \tag{14}$$

Adversary $\mathcal{A}_{\rm se,1}$ starts by executing the steps of INIT in Figure 7. It then runs $\mathcal{A}_{\rm cs}$, itself replying to the latter's oracle queries, to get $\mathcal{A}_{\rm cs}$'s output bit c', and returns the latter as its own guess bit. It replies to PDATA and HASH queries of $\mathcal{A}_{\rm cs}$ as per Figure 7. For the VOUCH query, it proceeds as in game G_3, executing lines 1–8. At line 9, instead of picking ct_1', ct_2', it queries its own ENC oracle to get $ct_1' \leftarrow\!\!\$\ \textsc{Enc}(2i-1, (adct, sh))$ and $ct_2' \leftarrow\!\!\$\ \textsc{Enc}(2i, (adct, sh))$. (Here i is the For loop counter from line 2.) It then reverts to following game G_3 for lines 10–12, the response to the query being as per line 12. For the analysis, let $K_1, K_2, \ldots, K_{2m-1}, K_{2m}$ denote the keys chosen at line 3 of $\mathbf{G}^{\rm ind\$}_{\mathsf{SE},2m}$.INIT. Then K_{2i-1}, K_{2i} play the role of K_1', K_2' in game G_2. So if the challenge bit d of game $\mathbf{G}^{\rm ind\$}_{\mathsf{SE},2m}$ is 1 then $\mathcal{A}_{\rm cs}$ gets the view it would in game G_2, and if d is 0 then $\mathcal{A}_{\rm cs}$ gets the view it would in game G_3, which implies Equation (14).

Now, using Equation (14), we have

$$\Pr[G_2(\mathcal{A}_{\rm cs})] = \Pr[G_3(\mathcal{A}_{\rm cs})] + (\Pr[G_2(\mathcal{A}_{\rm cs})] - \Pr[G_3(\mathcal{A}_{\rm cs})]) \tag{15}$$

$$\le \Pr[G_3(\mathcal{A}_{\rm cs})] + \mathbf{Adv}^{\rm ind\$}_{\mathsf{SE},2m}(\mathcal{A}_{\rm se,1}) \ , \tag{16}$$

With G_3, we have shown that the adversary gets no information about y when $y \notin X$. Now our task is to bound $\Pr[G_3(\mathcal{A}_{\rm cs})]$ to show that what the adversary learns when $y \in X$ is only what is permitted by the output. We break this into two cases, the first being that the number of matches is below threshold, the second being that it is above. Consider games G_4, G_5 of Figure 10, which capture these two cases, respectively. The games differ only in that line 2 is in G_4 alone and line 3

```
VOUCH($\mathbf{t}_0, \mathbf{t}_1$): // Games $G_4, G_5$
out $\leftarrow$ Out($\mathbf{t}_0, \mathbf{t}_1, X$) ; $s \leftarrow |\mathrm{ID}(\mathbf{t}_0, X)|$
If ($s \geq t+1$) then return $\perp$   // Line included only in game $G_4$
If ($s \leq t$) then return $\perp$   // Line included only in game $G_5$
For $i = 1, \ldots, |\mathbf{t}_0|$ do
    $(y, id, ad) \leftarrow \mathbf{t}_c[i]$
    $Q_1 \leftarrow \beta_1 R + \gamma_1 G$ ; $Q_2 \leftarrow \beta_2 R + \gamma_2 G$
    $ct'_1 \leftarrow_{\$} \{0,1\}^{\ell+e}$ ; $ct'_2 \leftarrow_{\$} \{0,1\}^{\ell+e}$
    If $y \notin X$ then $\mathbf{v}[i] \leftarrow (id, Q_1, ct'_1, Q_2, ct'_2)$
    Else
        $adct \leftarrow_{\$} \mathsf{SE.Enc}(adkey, ad)$ ; $x \leftarrow_{\$} \mathbb{F} \setminus \{0\}$ ; $z \leftarrow \mathbf{P}_{adkey, a_1, \ldots, a_t}(x)$ ; $sh \leftarrow (x, z)$
        $R \leftarrow \mathrm{HASH}(y)$ ; $w_1 \leftarrow h_1(y)$ ; $w_2 \leftarrow h_2(y)$ ; $\beta_1, \beta_2, \gamma_1, \gamma_2 \leftarrow_{\$} \mathbb{Z}_q$ ; $b \leftarrow_{\$} \{1,2\}$
        $S_1 \leftarrow \beta_1 P_{w_b} + \gamma_1 L$ ; $S_2 \leftarrow \beta_2 P_{w_{3-b}} + \gamma_2 L$
        $K_1 \leftarrow \mathsf{KDF}(S_1)$ ; $K_2 \leftarrow \mathsf{KDF}(S_2)$ ; $ct_1 \leftarrow_{\$} \mathsf{SE.Enc}(K_1, (adct, sh))$ ; $ct_2 \leftarrow_{\$} \mathsf{SE.Enc}(K_2, (adct, sh))$
        $\mathbf{v}[i] \leftarrow (id, Q_1, ct_1, Q_2, ct_2)$
Return $\mathbf{v}$, out
```

Figure 10: Games G_4, G_5 for proof of Theorem 8.1.

is in G_5 alone. Beyond the games are the same as G_3 except for some restructuring. The purpose of the latter is to highlight that encryption under *adkey*, now at line 10, is only done when $y \in X$. This results in fewer ENC queries for the $\mathcal{A}_{\mathrm{se},2}$ we will construct later, which improves the concrete security (tightness) considerably. For now, let p be the probability that $s \leq t$, where s is computed at line 1. This is the probability that the number of matches is below threshold. So we have

$$\Pr[G_3(\mathcal{A}_{\mathrm{cs}})] = p \cdot \Pr[G_4(\mathcal{A}_{\mathrm{cs}})] + (1-p) \cdot \Pr[G_5(\mathcal{A}_{\mathrm{cs}})] \ . \tag{17}$$

We turn to bounding $\Pr[G_4(\mathcal{A}_{\mathrm{cs}})]$ and $\Pr[G_5(\mathcal{A}_{\mathrm{cs}})]$.

Since the number of matches s in game G_4 is below threshold, the shares of *adkey* that $\mathcal{A}_{\mathrm{cs}}$ acquires give it no information about *adkey*. This allows us to move to game G_6 of Figure 11, where at line 10, the key used as the secret for the Shamir secret sharing, rather than being *adkey*, is a fresh, different key sampled at line 3. The perfect privacy of Shamir secret sharing implies that

$$\Pr[G_4(\mathcal{A}_{\mathrm{cs}})] = \Pr[G_6(\mathcal{A}_{\mathrm{cs}})] \ . \tag{18}$$

This move allows us to exploit the assumed IND\$-security of SE to replace *adct* by a random string. This is the change made in moving to G_7 of Figure 11, showing up at line 10 of the latter. Here a denotes the length of the message *ad* that is being encrypted, and e is the ciphertext-length overhead of SE. (We are again using the assumption that the length of the associated data is a fixed, known constant across all queries, which means that the length of the messages being encrypted is

```
$\textsc{Vouch}(\mathbf{t}_0, \mathbf{t}_1)$:  // Game $G_6$
out $\leftarrow$ Out$(\mathbf{t}_0, \mathbf{t}_1, X)$ ; $s \leftarrow |\mathrm{ID}(\mathbf{t}_0, X)|$
If $(s \geq t+1)$ then return $\bot$
$adkey' \leftarrow\$ \{0,1\}^k$
For $i = 1, \ldots, |\mathbf{t}_0|$ do
    $(y, id, ad) \leftarrow \mathbf{t}_c[i]$
    $Q_1 \leftarrow \beta_1 R + \gamma_1 G$ ; $Q_2 \leftarrow \beta_2 R + \gamma_2 G$
    $ct_1' \leftarrow\$ \{0,1\}^{\ell+e}$ ; $ct_2' \leftarrow\$ \{0,1\}^{\ell+e}$
    If $y \notin X$ then $\mathbf{v}[i] \leftarrow (id, Q_1, ct_1', Q_2, ct_2')$
    Else
        $adct \leftarrow\$ \mathsf{SE.Enc}(adkey, ad)$ ; $x \leftarrow\$ \mathbb{F} \setminus \{0\}$ ; $z \leftarrow \mathbf{P}_{adkey', a_1, \ldots, a_t}(x)$ ; $sh \leftarrow (x, z)$
        $R \leftarrow \textsc{Hash}(y)$ ; $w_1 \leftarrow h_1(y)$ ; $w_2 \leftarrow h_2(y)$ ; $\beta_1, \beta_2, \gamma_1, \gamma_2 \leftarrow\$ \mathbb{Z}_q$ ; $b \leftarrow\$ \{1,2\}$
        $S_1 \leftarrow \beta_1 P_{w_b} + \gamma_1 L$ ; $S_2 \leftarrow \beta_2 P_{w_{3-b}} + \gamma_2 L$
        $K_1 \leftarrow \mathsf{KDF}(S_1)$ ; $K_2 \leftarrow \mathsf{KDF}(S_2)$ ; $ct_1 \leftarrow\$ \mathsf{SE.Enc}(K_1, (adct, sh))$ ; $ct_2 \leftarrow\$ \mathsf{SE.Enc}(K_2, (adct, sh))$
        $\mathbf{v}[i] \leftarrow (id, Q_1, ct_1, Q_2, ct_2)$
Return $\mathbf{v}$, out
```

```
$\textsc{Vouch}(\mathbf{t}_0, \mathbf{t}_1)$:  // Game $G_7$
out $\leftarrow$ Out$(\mathbf{t}_0, \mathbf{t}_1, X)$ ; $s \leftarrow |\mathrm{ID}(\mathbf{t}_0, X)|$
If $(s \geq t+1)$ then return $\bot$
$adkey' \leftarrow\$ \{0,1\}^k$
For $i = 1, \ldots, |\mathbf{t}_0|$ do
    $(y, id, ad) \leftarrow \mathbf{t}_c[i]$
    $Q_1 \leftarrow \beta_1 R + \gamma_1 G$ ; $Q_2 \leftarrow \beta_2 R + \gamma_2 G$
    $ct_1' \leftarrow\$ \{0,1\}^{\ell+e}$ ; $ct_2' \leftarrow\$ \{0,1\}^{\ell+e}$
    If $y \notin X$ then $\mathbf{v}[i] \leftarrow (id, Q_1, ct_1', Q_2, ct_2')$
    Else
        $adct \leftarrow\$ \{0,1\}^{a+e}$ ; $x \leftarrow\$ \mathbb{F} \setminus \{0\}$ ; $z \leftarrow \mathbf{P}_{adkey', a_1, \ldots, a_t}(x)$ ; $sh \leftarrow (x, z)$
        $R \leftarrow \textsc{Hash}(y)$ ; $w_1 \leftarrow h_1(y)$ ; $w_2 \leftarrow h_2(y)$ ; $\beta_1, \beta_2, \gamma_1, \gamma_2 \leftarrow\$ \mathbb{Z}_q$ ; $b \leftarrow\$ \{1,2\}$
        $S_1 \leftarrow \beta_1 P_{w_b} + \gamma_1 L$ ; $S_2 \leftarrow \beta_2 P_{w_{3-b}} + \gamma_2 L$
        $K_1 \leftarrow \mathsf{KDF}(S_1)$ ; $K_2 \leftarrow \mathsf{KDF}(S_2)$ ; $ct_1 \leftarrow\$ \mathsf{SE.Enc}(K_1, (adct, sh))$ ; $ct_2 \leftarrow\$ \mathsf{SE.Enc}(K_2, (adct, sh))$
        $\mathbf{v}[i] \leftarrow (id, Q_1, ct_1, Q_2, ct_2)$
Return $\mathbf{v}$, out
```

Figure 11: Games G_6, G_7 for proof of Theorem 8.1.

also the same across all queries.) We construct adversary $\mathcal{A}_{\mathrm{se},2}$, playing game $\mathbf{G}^{\mathrm{ind\$}}_{\mathsf{SE},1}$, so that

$$\Pr[G_6(\mathcal{A}_{\mathrm{cs}})] - \Pr[G_7(\mathcal{A}_{\mathrm{cs}})] \leq \mathbf{Adv}^{\mathrm{ind\$}}_{\mathsf{SE},1}(\mathcal{A}_{\mathrm{se},2}) \; . \tag{19}$$

Adversary $\mathcal{A}_{\mathrm{se},2}$ starts by executing the steps of INIT in Figure 7. It then runs $\mathcal{A}_{\mathrm{cs}}$, itself replying to the latter's oracle queries, to get $\mathcal{A}_{\mathrm{cs}}$'s output bit c', and returns the latter as its own guess bit. It replies to PDATA and HASH queries of $\mathcal{A}_{\mathrm{cs}}$ as per Figure 7. For the VOUCH query, it proceeds as in game G_7, executing lines 1–9. At line 10, instead of picking *adct*, it queries its own ENC oracle to get $adct \leftarrow\$ \,\mathrm{ENC}(1, ad)$. It then reverts to following game G_7, for the rest of line 10 and for lines 11–14, the response to the query being as per line 15. For the analysis, let *adkey* denote the key chosen at line 3 of $\mathbf{G}^{\mathrm{ind\$}}_{\mathsf{SE},1}$.INIT. Then *adkey* plays the role of *adkey* in game G_6. So if the challenge bit d of game $\mathbf{G}^{\mathrm{ind\$}}_{\mathsf{SE},1}$ is 1 then $\mathcal{A}_{\mathrm{cs}}$ gets the view it would in game G_6, and if d is 0 then $\mathcal{A}_{\mathrm{cs}}$ gets the view it would in game G_7, which implies Equation (19). The number of ENC queries of $\mathcal{A}_{\mathrm{se},2}$ is $s = |\mathrm{ID}(\mathbf{t}_0, X)|$, which by assumption is below μ, and by line 2 of G_6 is also below t, so is below $\min(\mu, t)$ as claimed.

At this point, the messages encrypted in ct_1, ct_2 no longer depend on the associated data, so the adversary gets no information about the latter. Now we claim that

$$\Pr[\mathrm{G}_7(\mathcal{A}_{\mathrm{cs}})] = \frac{1}{2}\ , \tag{20}$$

meaning the adversary in G_7 gets no information about the challenge bit c. To show this, we need to show that the adversary's view is independent of the y determined at line 5. The query response is clearly independent of y when $y \notin X$ (this is what prior games have accomplished) so suppose $y \in X$. In this case, $\alpha R \in \{P_{w_1}, P_{w_2}\}$. The assumption that $\mathrm{Valid}(pdata) = \mathtt{true}$ means, first, that $w_1 \neq w_2$ and second that $P_{w_1} \neq P_{w_2}$, so let $j \in \{1, 2\}$ be such that $\alpha R = P_{w_j} \neq P_{w_{3-j}}$. Then Lemma 8.2 part (2) tells us how (Q_j, S_j) is distributed and Lemma 8.2 part (1) tells us how (Q_{3-j}, S_{3-j}) is distributed. These distributions do not depend on y. Finally, due to the random choice of b at line 7, j itself does not depend on y. So the only thing the adversary learns is whether or not $y \in X$, but the definition of function Out in Figure 3 ensures that the two vectors queried to VOUCH agree in this regard. This justifies Equation (20).

Next we return to the case that the number of matches is above threshold and claim that

$$\Pr[\mathrm{G}_5(\mathcal{A}_{\mathrm{cs}})] = \frac{1}{2}\ , \tag{21}$$

In this case, enough Shamir shares are available that the adversary recovers *adkey* and *adct*, and thus *ad*. However, the definition of function Out in Figure 3 ensures that the two vectors queried to VOUCH agree in whatever information the adversary may recover.

Recalling how $\epsilon_{\mathrm{cs}}, \epsilon_{\mathrm{kdf}}, \epsilon_{\mathrm{se},1}, \epsilon_{\mathrm{se},2}$ were defined in the theorem statement, we can now put the above equations together to get

$$\begin{aligned}\frac{\epsilon_{\mathrm{cs}}}{2}+\frac{1}{2} &\le \epsilon_{\mathrm{kdf}}+\epsilon_{\mathrm{se},1}+p\cdot\left(\frac{1}{2}+\epsilon_{\mathrm{se},2}\right)+(1-p)\cdot\frac{1}{2}+\frac{2(m+h)}{q-1}\\ &\le \epsilon_{\mathrm{kdf}}+\epsilon_{\mathrm{se},1}+\epsilon_{\mathrm{se},2}+\frac{1}{2}+\frac{2(m+h)}{q-1}\ . \qquad (22)\end{aligned}$$

Rearranging terms yields Equation (6).

Recall that, as per our conventions, the running time of $\mathcal{A}_{\mathrm{cs}}$ means that of the execution of $\mathcal{A}_{\mathrm{cs}}$ with game $\mathbf{G}^{\mathrm{cs}}_{\mathsf{\Pi},\mathsf{Ex},\mathsf{Out}}$. This includes the time taken by game procedures (oracles) to compute their replies to queries, which in turn includes the time to run Ex. With this convention, the running time of the constructed adversaries is indeed essentially that of $\mathcal{A}_{\mathrm{cs}}$ as claimed in the theorem statement. ▮

9 Protocol correctness

Correctness refers to the property that a protocol achieves the desired functionality, meaning, on any inputs, returns the desired output. (As an analogy, correctness of an encryption scheme asks that decryption reverse encryption.) Usually, this is simple, and may not be treated beyond a one-line claim. Correctness in the Apple protocol, however, is more involved. This is in part because it relies on properties of the underlying cryptographic primitives. Most importantly, it relies on a robustness property of the symmetric encryption scheme. It also relies on the security of the KDF, and on certain parameters being large enough. In cryptographic parlance, this makes correctness, here, a computational property, as opposed to the statistical one it is usually.

Treating it, accordingly, requires that we first step back to definitions. We define correctness in the same style as we have defined security, via a game, and then an advantage, associated to an adversary playing that game. Then we give a theorem that upper bounds this adversary advantage as a function of the advantage of other adversaries, that we construct, that aim to violate security properties of the primitives. The bound also includes terms involving the size of the group $\mathbb{G}$ and the field $\mathbb{F}$. For appropriate parameter choices, this bound is small enough, which we take as establishing correctness of the protocol.

The correctness game $\mathbf{G}^{\mathrm{corr}}_{\mathsf{\Pi}}$ that we associate to protocol $\mathsf{\Pi}$ is shown in Figure 12. FIN returns true when correctness fails, so that the advantage of adversary $\mathcal{A}$, defined as

$$\mathbf{Adv}^{\mathrm{corr}}_{\mathsf{\Pi}}(\mathcal{A})=\Pr[\mathbf{G}^{\mathrm{corr}}_{\mathsf{\Pi}}(\mathcal{A})]\ , \qquad (23)$$

is its probability of violating correctness. The game begins, as usual, by picking the function H that will play the role of the random oracle via procedure HASH. The adversary begins by querying a set X to procedure POST. (It is required to make exactly one query to this oracle.) The game runs the protocol algorithm SePost on input X and returns the resulting *pdata* to the adversary.

```
Game $\mathbf{G}^{\mathrm{corr}}_{\Pi}$

INIT:
1  $\mathrm{H} \leftarrow\!\!\$\ \Pi.\mathsf{HS}$   // Pick a function to be the random oracle

POST($X$):  // Adversary calls with set $X$.
2  Require: $X \subseteq \Pi.\mathsf{U}$
3  $(pdata, skey) \leftarrow\!\!\$\ \Pi.\mathsf{SePost}^{\textsc{Hash}}(X)$
4  Return $pdata$   // Adversary is given the public data

VOUCH($\mathbf{t}$):  // Adversary calls with a vector of triples
5  $m \leftarrow |\mathbf{t}|$
6  Require: $\mathrm{ID}(\mathbf{t}[1]), \ldots, \mathrm{ID}(\mathbf{t}[m])$ are all distinct
7  $ckey \leftarrow\!\!\$\ \mathsf{ClInit}[\textsc{Hash}](pdata)$
8  For $i = 1, \ldots, m$ do
9      $\mathbf{v}[i] \leftarrow\!\!\$\ \mathsf{ClVch}[\textsc{Hash}](pdata, ckey, \mathbf{t}[i])$ ; $sList \leftarrow \mathsf{SeCollect}(pdata, skey, \mathbf{v}[i], sList)$
10 $sout \leftarrow \mathsf{SeOut}[\textsc{Hash}](pdata, skey, sList)$
11 Return $\perp$

HASH($W$):  // The random oracle
12 Return $\mathrm{H}(W)$

FIN():  //
13 If (not Valid($pdata$)) then return true   // Correctness fails, adversary wins
14 $\phi \leftarrow \mathcal{F}(\mathbf{t}, X)$
15 Return $(sout \neq \phi)$   // Result of game, true if $sout \neq \phi$ and false otherwise
```

```
Function $\mathcal{F}(\mathbf{t}, X)$
$m \leftarrow |\mathbf{t}|$
$s \leftarrow |\mathrm{ID}(\mathbf{t}, X)|$   // Number of matches
If $s \leq t$ then   // Number of matches is below threshold
    out $\leftarrow (\mathrm{ID}(\mathbf{t}), \mathrm{Match}(\mathbf{t}, X))$   // Output the locations of the matches
Else   // $s \geq t + 1$, number of matches is above threshold
    out $\leftarrow (\mathrm{ID}(\mathbf{t}), \mathrm{AD}(\mathbf{t}, X))$   // Output the identifiers and associated data of the matches
Return out
```

Figure 12: Game defining correctness of protocol Π.

Next the adversary calls VOUCH with a vector $\mathbf{y}$ of triples, which the game processes according to the protocol, returning nothing (formally, $\perp$) to the adversary. At line 13, procedure FIN returns true (correctness failure) if *pdata* fails the validity condition. Else, at line 14, it sets ϕ to the

```
Game $\mathbf{G}^{\mathrm{rob\$}}_{\mathsf{SE},s,r}$

INIT:
  For $i = 1, \ldots, s$ do $K_i \leftarrow\!\!\$\ \{0,1\}^k$
  For $i = j, \ldots, r$ do $\overline{K}_j \leftarrow\!\!\$\ \{0,1\}^k$

ENC$(i, M)$:  // Adversary calls with sender $i \in [1..s]$ and message $M$
  Require: $1 \le i \le s$
  $w \leftarrow w + 1$ ; $C_w \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_i, M)$
  Return $\perp$

DEC$(j, l)$:  // Adversary calls with receiver $j \in [1..r]$ and index $l \in [1..w]$
  Require: $1 \le j \le r$ and $1 \le l \le w$
  $M \leftarrow \mathsf{SE.Dec}(\overline{K}_j, C_l)$
  If $(M \neq \perp)$ then win $\leftarrow$ true
  Return $\perp$

FIN:  // No adversary input
  Return win
```

Figure 13: Game defining ROB\$ security of SE.

desired protocol output as determined by the functionality $\mathcal{F}$ defined at the bottom of the same Figure. The adversary wins (the game returns true) if the protocol output *sout* is different from the functionality output ϕ.

The random oracle HASH is called by scheme algorithms invoked in game procedures, but we disallow the adversary from calling it, meaning we assume $\mathcal{A}$ makes zero HASH queries. This reflects the expectation that protocol inputs are not dependent on the random oracle.

Robustness of an encryption scheme, a concept introduced in [3], asks that decryption under the wrong key fails. That is, if $C \leftarrow\!\!\$\ \mathsf{SE.Enc}(K, M)$ and $M' \leftarrow \mathsf{SE.Dec}(K', C)$ for $K' \neq K$, then M' should be $\perp$. The definition has many variants depending on the allowed choices of K, K', C, and has been further explored in [3, 14, 15, 16], the last paper using the term committing in place of robust. Correctness of the Apple protocol relies on a form of robustness, defined in [11], that we call random-key robustness. Its merit is that it is weak enough that, unlike stronger forms of robustness, it is implied by the standard authenticity property of an authenticated encryption scheme. This can be shown directly or obtained as a consequence of [15, Theorem 2].

We give a definition of random-key robustness, that we call ROB\$, that extends the definition of [11] to a multi-user setting. (This allows a tight reduction in Theorem 9.1.) The definition is based on game $\mathbf{G}^{\mathrm{rob\$}}_{\mathsf{SE},s,r}$ of Figure 13. If $\mathcal{A}_{\mathrm{rob}}$ is an adversary playing this game, its advantage is

$$\mathbf{Adv}^{\mathrm{rob\$}}_{\mathsf{SE},s,r}(\mathcal{A}_{\mathrm{rob}}) = \Pr[\mathbf{G}^{\mathrm{rob\$}}_{\mathsf{SE},s,r}(\mathcal{A}_{\mathrm{rob}})] \ . \tag{24}$$

Here SE is the symmetric encryption scheme whose robustness is being defined, integer $s \geq 1$ represents the number of senders and integer $r \geq 1$ represents the number of receivers. Lines 1,2 pick keys for all senders and receivers. The adversary can call oracle $\textsc{Enc}$ with the identity $i \in [1..s]$ of a sender and a message M to have the game create a ciphertext encrypting M under the key K_i of sender i. It gets back $\perp$ in response to the call. (It does not get back the ciphertext.) It can also call $\textsc{Dec}$, giving a receiver identity j and a pointer l indicating a previously-created ciphertext C_l. Oracle $\textsc{Dec}$ decrypts C_l under the key $\overline{K}_j$ of receiver j. (Again, the response returned to the adversary query is $\perp$.) The adversary wins if the result of this decryption is not $\perp$, meaning is a valid message. The game returns true if the adversary submits some winning $\textsc{Dec}$ query.

We need ultimately to rely on the authenticity of SE, so the tightness of the reduction from authenticity to ROB\$ is important. In Appendix A, we confirm that it is indeed tight, in the multi-user setting.

The following theorem establishes correctness of the protocol assuming ROB\$-security of SE, security of the KDF, and that the sizes of the group and the field are large enough.

Theorem 9.1 Let Π be the basic Apple protocol (Figure 1) associated to the ingredients listed in Section 4, and in particular let $q = |\mathbb{G}|$ be the size of the group and $f = |\mathbb{F}|$ the size of the field used in the Shamir secret sharing. Let $\mathcal{A}$ be an adversary, playing game $\mathbf{G}^{\mathrm{corr}}_{\Pi}$, that makes one $\textsc{Post}$ query, and let n' be the size of the table created by MkHT_1 given the size n of the set X in this query. Let $\epsilon_{\mathrm{corr}} = \mathbf{Adv}^{\mathrm{corr}}_{\Pi}(\mathcal{A})$ be the advantage of $\mathcal{A}$ in violating correctness. Assume $\mathcal{A}$ makes no $\textsc{Hash}$ queries, and let m be the length of the vector in its (one) $\textsc{Vouch}$ query. Then we can construct adversaries $\mathcal{A}_{\mathrm{kdf}}, \mathcal{A}_{\mathrm{rob}}$ such that

$$\epsilon_{\mathrm{corr}} \leq \epsilon_{\mathrm{kdf}} + \epsilon_{\mathrm{rob}} + \frac{4m + n'(n'-1)}{2(q-1)} + \frac{m(m-1)}{2(f-1)} \ , \tag{25}$$

where:

- $\epsilon_{\mathrm{kdf}} = \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{KDF},4m}(\mathcal{A}_{\mathrm{kdf}})$ is the advantage of $\mathcal{A}_{\mathrm{kdf}}$. It is playing game $\mathbf{G}^{\mathrm{kdf}}_{\mathsf{KDF},4m}$.
- $\epsilon_{\mathrm{rob}} = \mathbf{Adv}^{\mathrm{rob\$}}_{\mathsf{SE},2m,2m}(\mathcal{A}_{\mathrm{rob}})$ is the advantage of $\mathcal{A}_{\mathrm{rob}}$. It is playing game $\mathbf{G}^{\mathrm{rob\$}}_{\mathsf{SE},2m,2m}$. It makes one query, per each of its $2m$ senders, to its $\textsc{Enc}$ oracle, and the analogously for its $\textsc{Dec}$ oracle.

The constructed adversaries all have about the same running time as $\mathcal{A}$. ▌

The proof needs the following analogue of Lemma 8.2.

Lemma 9.2 Let $\mathbb{G}$ be a group of prime order q, and G a generator of $\mathbb{G}$. Let $\alpha \in \mathbb{Z}_q^*$ and let $L = \alpha G$. Let $P, R \in \mathbb{G} \setminus \{0\}$. Define $\mathsf{Q}, \mathsf{S}\colon \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ by $\mathsf{Q}(\beta,\gamma) = \beta R + \gamma G$ and $\mathsf{S}(\beta,\gamma) = \beta P + \gamma L$. Then define $\overline{\mathsf{S}}\colon \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ by $\overline{\mathsf{S}}(\beta,\gamma) = \alpha\, \mathsf{Q}(\beta,\gamma)$. Regard $\mathsf{Q}, \mathsf{S}, \overline{\mathsf{S}}$ as random variables over the random choices $\beta, \gamma \leftarrow\$\, \mathbb{Z}_q$. Assume $P \neq \alpha R$. Then the pair $(\overline{\mathsf{S}}, \mathsf{S})$ is uniformly distributed over $\mathbb{G} \times \mathbb{G}$.

Proof of Lemma 9.2: Let $r \leftarrow \mathrm{DLog}_{\mathbb{G},G}(R)$ and $p \leftarrow \mathrm{DLog}_{\mathbb{G},G}(P)$. Let $t \leftarrow \mathrm{DLog}_{\mathbb{G},G}(\overline{\mathsf{S}}(\beta,\gamma))$ and $s \leftarrow \mathrm{DLog}_{\mathbb{G},G}(\mathsf{S}(\beta,\gamma))$. Then

$$\begin{bmatrix} r\alpha & \alpha \\ p & \alpha \end{bmatrix} \cdot \begin{bmatrix} \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} t \\ s \end{bmatrix} .$$

The determinant of the 2 by 2 matrix on the left is $D = (r\alpha^2 - p\alpha) \bmod q = \alpha(r\alpha - p) \bmod q$. The assumptions that $\alpha \neq 0$ and $P \neq \alpha\, R$ mean that $D \neq 0$, so the matrix is invertible. So for every choice of $(t,s) \in \mathbb{Z}_q^2$ there is a unique $(\beta,\gamma) \in \mathbb{Z}_q^2$ such that the above matrix equation is true. The claim follows. ▌

Proof of Theorem 9.1: Game G_0 of Figure 14 excludes the boxed code, while game G_1 includes it. We claim that

$$\mathbf{Adv}^{\mathrm{corr}}_{\Pi}(\mathcal{A}) \leq \Pr[\mathrm{G}_0(\mathcal{A})] + \Pr[\mathrm{G}_0(\mathcal{A}) \text{ sets } \mathsf{bad}] \; . \tag{26}$$

To justify this, consider the ways in which correctness fails, meaning the adversary wins in game $\mathbf{G}^{\mathrm{corr}}_{\Pi}$. The adversary wins if $\mathrm{Valid}(pdata) = \mathsf{false}$. This comes down to $P_1, \ldots, P_{n'}$ not being distinct, for $\mathrm{MkHT}_1, \mathrm{MkHT}_2$ assure the other conditions are met. We capture this by having lines 8,12 of game G_0 set bad if $P_1, \ldots, P_{n'}$ are not distinct. The recovery procedure of Shamir secret sharing may fail if there are shares with the same first component, captured by setting bad at line 19. Next, if y at line 18 is not in X, the correctness fails if either of the decryptions at line 25 is not $\perp$, in which case bad_1 is set. If $y \in X$, one decryption will be correct, and correctness fails if the other returns a non-$\perp$ result, captured by setting bad_2 at line 29. The game returns true if either bad_1 or bad_2 was set.

At line 8, bad is set with probability at most $(i-1)/(q-1)$, where i is the loop counter from line 4, and similarly for line 12. At line 19, bad is set with probability at most $(i-1)/(f-1)$ where i is the loop counter from line 17. So overall

$$\begin{aligned} \Pr[\mathrm{G}_0(\mathcal{A}) \text{ sets } \mathsf{bad}] &\leq \sum_{i=1}^{n'} \frac{i-1}{q-1} + \sum_{i=1}^{m} \frac{i-1}{f-1} \\ &\leq \frac{n'(n'-1)}{2(q-1)} + \frac{m(m-1)}{2(f-1)} \; . \end{aligned} \tag{27}$$

Games $\mathrm{G}_0, \mathrm{G}_1$ are identical-until-bad, and also return true (line 33) only under the condition that $\mathsf{bad} = \mathsf{true}$. A variant of the Fundamental Lemma of Game Playing of [8], stated in [5], then says that

$$\Pr[\mathrm{G}_0(\mathcal{A})] = \Pr[\mathrm{G}_1(\mathcal{A})] \; . \tag{28}$$

We turn to bounding $\Pr[\mathrm{G}_1(\mathcal{A})]$. All further games will use the procedures in Figure 15; they will

```
Games G_0, \boxed{G_1}

INIT:
$adkey \leftarrow\!\!\$ \{0,1\}^k$ ; $a_1,\ldots,a_t \leftarrow\!\!\$ \mathbb{F}$ ; $ckey \leftarrow (adkey, a_1,\ldots,a_t)$ ; $\alpha \leftarrow\!\!\$ \mathbb{Z}_q^*$

POST(X):  // Adversary calls with set X.
Require: $X \subseteq \Pi.\mathsf{U}$
$n \leftarrow |X|$ ; $(h_1,h_2,n') \leftarrow\!\!\$ \mathrm{MkHT}_1(n)$ ; $T \leftarrow \mathrm{MkHT}_2(X,h_1,h_2,n')$ ; $\mathrm{XH} \leftarrow \{0\}$ ; $\mathrm{PS} \leftarrow \{0\}$ ; $\mathrm{XS} \leftarrow \{0\}$
For $i = 1,\ldots,n'$ do
    $x \leftarrow T[i]$
    If $(x \neq \bot)$ then
        $\mathrm{HT}[x] \leftarrow\!\!\$ \mathbb{G}\setminus\{0\}$
        If $(\mathrm{HT}[x] \in \mathrm{XH})$ then $\mathsf{bad} \leftarrow \mathsf{true}$ ; $\boxed{\mathrm{HT}[x] \leftarrow\!\!\$ \mathbb{G}\setminus \mathrm{XH}}$
        $\mathrm{XH} \leftarrow \mathrm{XH}\cup\{\mathrm{HT}[x]\}$ ; $P_i \leftarrow \alpha\,\mathrm{HT}[x]$
    Else
        $P_i \leftarrow\!\!\$ \mathbb{G}\setminus\{0\}$
        If $(P_i \in \mathrm{PS})$ then $\mathsf{bad} \leftarrow \mathsf{true}$ ; $\boxed{P_i \leftarrow\!\!\$ \mathbb{G}\setminus \mathrm{PS}}$
    $\mathrm{PS} \leftarrow \mathrm{PS}\cup\{P_i\}$
$pdata \leftarrow (L, n', P_1,\ldots,P_{n'}, h_1, h_2)$ ; Return $pdata$

VOUCH(t):  // Adversary calls with a vector of triples
$m \leftarrow |\mathbf{t}|$
Require: $\mathrm{ID}(\mathbf{t}[1]),\ldots,\mathrm{ID}(\mathbf{t}[m])$ are all distinct
For $i = 1,\ldots,|\mathbf{t}|$ do
    $(y, id, ad) \leftarrow \mathbf{t}[i]$ ; $adct \leftarrow\!\!\$ \mathsf{SE.Enc}(adkey, ad)$ ; $x \leftarrow\!\!\$ \mathbb{F}\setminus\{0\}$
    If $(x \in \mathrm{XS})$ then $\mathsf{bad} \leftarrow \mathsf{true}$ ; $\boxed{x \leftarrow\!\!\$ \mathbb{F}\setminus \mathrm{XS}}$
    $\mathrm{XS} \leftarrow \mathrm{XS}\cup\{x\}$ ; $z \leftarrow \mathbf{P}_{adkey,a_1,\ldots,a_t}(x)$ ; $sh \leftarrow (x,z)$
    $R \leftarrow \mathrm{HASH}(y)$ ; $w_1 \leftarrow h_1(y)$ ; $w_2 \leftarrow h_2(y)$ ; $\beta_1,\beta_2,\gamma_1,\gamma_2 \leftarrow\!\!\$ \mathbb{Z}_q$ ; $b \leftarrow\!\!\$ \{1,2\}$
    $Q_1 \leftarrow \beta_1 R + \gamma_1 G$ ; $Q_2 \leftarrow \beta_2 R + \gamma_2 G$ ; $S_1 \leftarrow \beta_1 P_{w_b} + \gamma_1 L$ ; $S_2 \leftarrow \beta_2 P_{w_{3-b}} + \gamma_2 L$
    $K_1 \leftarrow \mathsf{KDF}(S_1)$ ; $K_2 \leftarrow \mathsf{KDF}(S_2)$ ; $ct_1 \leftarrow\!\!\$ \mathsf{SE.Enc}(K_1,(adct,sh))$ ; $ct_2 \leftarrow\!\!\$ \mathsf{SE.Enc}(K_2,(adct,sh))$
    $S_1' \leftarrow \alpha Q_1$ ; $S_2' \leftarrow \alpha Q_2$
    $K_1' \leftarrow \mathsf{KDF}(S_1')$ ; $K_2' \leftarrow \mathsf{KDF}(S_2')$ ; $M_1 \leftarrow \mathsf{SE.Dec}(K_1', ct_1)$ ; $M_2 \leftarrow \mathsf{SE.Dec}(K_2', ct_2)$
    If $(y \notin X$ and $(M_1,M_2) \neq (\bot,\bot))$ then $\mathsf{bad}_1 \leftarrow \mathsf{true}$
    If $y \in X$ then
        If $(R = P_{w_b})$ then $j \leftarrow 2$ else $j \leftarrow 1$
        If $(M_j \neq \bot)$ then $\mathsf{bad}_2 \leftarrow \mathsf{true}$
Return $\bot$

HASH(y):  // The random oracle
If $\mathrm{HT}[y] = \bot$ then $\mathrm{HT}[y] \leftarrow\!\!\$ \mathbb{G}\setminus\{0\}$
Return $\mathrm{HT}[y]$

FIN():  //
Return $((\mathsf{bad}_1 \vee \mathsf{bad}_2) \wedge (\text{not } \mathsf{bad}))$
```

Figure 14: Games G_0, G_1 for proof of Theorem 9.1, where the latter includes the boxed code and the former does not.

```
Games G_2–G_5

INIT:
adkey ←$ {0,1}^k ; a_1, ..., a_t ←$ F ; ckey ← (adkey, a_1, ..., a_t) ; α ←$ Z_q^*

POST(X):  // Adversary calls with set X.
Require: X ⊆ Π.U
n ← |X| ; (h_1, h_2, n') ←$ MkHT_1(n) ; T ← MkHT_2(X, h_1, h_2, n') ; XH ← {0} ; PS ← {0} ; XS ← {0}
For i = 1, ..., n' do
    x ← T[i]
    If (x ≠ ⊥) then HT[x] ←$ G \ XH ; XH ← XH ∪ {HT[x]} ; P_i ← α HT[x]
    Else P_i ←$ G \ PS
    PS ← PS ∪ {P_i}
pdata ← (L, n', P_1, ..., P_{n'}, h_1, h_2) ; Return pdata

HASH(y):  // The random oracle
If HT[y] = ⊥ then HT[y] ←$ G \ {0}
Return HT[y]

FIN():  //
Return (bad_1 ∨ ··· ∨ bad_m)
```

Figure 15: Procedures used by games G_2–G_5 for proof of Theorem 9.1.

differ only in their VOUCH procedures, which will be specified individually, and which set the flags $\mathsf{bad}_1, \ldots, \mathsf{bad}_m$ referred to at line 12 of Figure 15. In Figure 15, $P_1, \ldots, P_{n'}$ are chosen, as per game G_1, to be distinct.

Consider game G_2 of Figure 16. We claim that

$$\Pr[G_1(\mathcal{A})] = \Pr[G_2(\mathcal{A})] . \tag{29}$$

We proceed to justify Equation (29). Game G_1, at line 24, had set S_1', S_2' as the server would in $\mathsf{SeCollect}$. Towards exploiting the assumed security of KDF, game G_2 of Figure 16, additionally and optimistically, choses $\overline{S}_1, \overline{S}_2$ at random at line 9. The claim is that, due to Lemma 9.2, these can correctly replace S_1', S_2', respectively, unless some "bad" event happens, in which case they are reverted to the latter values. We now expand on this. The optimistic choices $\overline{S}_1, \overline{S}_2$ are used in the decryptions of line 11 that define the messages $\overline{M}_1, \overline{M}_2$, and these are used to set bad_i at lines 14, 17. We consider separately the case $y \notin X$ and $y \in X$, beginning with the former. At line 13, if αR equals either P_{w_1} or P_{w_2}, then flag bad is set and the boxed code, which is included in game G_2, reverts the messages $\overline{M}_1, \overline{M}_2$ to their "correct" values of M_1, M_2, respectively. If bad is not set, however, we claim that substituting (S_1', S_2') by $(\overline{S}_1, \overline{S}_2)$ makes no difference. Indeed, since $\alpha R \neq P_{w_1}$, we can apply Lemma 9.2, with the randomness being β_1, γ_1, to conclude that (S_1, S_1')

```
Games $\boxed{\mathrm{G}_2}$, $\mathrm{G}_3$

$\textsc{Vouch}(\mathbf{t})$:  // Adversary calls with a vector of triples
$m \leftarrow |\mathbf{t}|$
Require: $\mathrm{ID}(\mathbf{t}[1]), \ldots, \mathrm{ID}(\mathbf{t}[m])$ are all distinct
For $i = 1, \ldots, |\mathbf{t}|$ do
    $(y, id, ad) \leftarrow \mathbf{t}[i]$ ; $adct \leftarrow\$ \mathsf{SE.Enc}(adkey, ad)$
    $x \leftarrow\$ \mathbb{F} \setminus \mathrm{XS}$; $\mathrm{XS} \leftarrow \mathrm{XS} \cup \{x\}$ ; $z \leftarrow \mathbf{P}_{adkey,a_1,\ldots,a_t}(x)$ ; $sh \leftarrow (x, z)$
    $R \leftarrow \textsc{Hash}(y)$ ; $w_1 \leftarrow h_1(y)$ ; $w_2 \leftarrow h_2(y)$ ; $\beta_1, \beta_2, \gamma_1, \gamma_2 \leftarrow\$ \mathbb{Z}_q$ ; $b \leftarrow\$ \{1, 2\}$
    $Q_1 \leftarrow \beta_1 R + \gamma_1 G$ ; $Q_2 \leftarrow \beta_2 R + \gamma_2 G$ ; $S_1 \leftarrow \beta_1 P_{w_b} + \gamma_1 L$ ; $S_2 \leftarrow \beta_2 P_{w_{3-b}} + \gamma_2 L$
    $K_1 \leftarrow \mathsf{KDF}(S_1)$ ; $K_2 \leftarrow \mathsf{KDF}(S_2)$ ; $ct_1 \leftarrow\$ \mathsf{SE.Enc}(K_1, (adct, sh))$ ; $ct_2 \leftarrow\$ \mathsf{SE.Enc}(K_2, (adct, sh))$
    $S_1' \leftarrow \alpha Q_1$ ; $S_2' \leftarrow \alpha Q_2$ ; $\overline{S}_1 \leftarrow\$ \mathbb{G}$ ; $\overline{S}_2 \leftarrow\$ \mathbb{G}$
    $K_1' \leftarrow \mathsf{KDF}(S_1')$ ; $K_2' \leftarrow \mathsf{KDF}(S_2')$ ; $M_1 \leftarrow \mathsf{SE.Dec}(K_1', ct_1)$ ; $M_2 \leftarrow \mathsf{SE.Dec}(K_2', ct_2)$
    $\overline{K}_1 \leftarrow \mathsf{KDF}(\overline{S}_1)$ ; $\overline{K}_2 \leftarrow \mathsf{KDF}(\overline{S}_2)$ ; $\overline{M}_1 \leftarrow \mathsf{SE.Dec}(\overline{K}_1, ct_1)$ ; $\overline{M}_2 \leftarrow \mathsf{SE.Dec}(\overline{K}_2, ct_2)$
    If $(y \notin X)$ then
        If $(\alpha R \in \{P_{w_1}, P_{w_2}\})$ then $\mathsf{bad} \leftarrow \mathsf{true}$ ; $\boxed{(\overline{M}_1, \overline{M}_2) \leftarrow (M_1, M_2)}$
        $\mathsf{bad}_i \leftarrow ((\overline{M}_1, \overline{M}_2) \neq (\bot, \bot))$
    If $y \in X$ then
        If $(R = P_{w_b})$ then $j \leftarrow 2$ else $j \leftarrow 1$
        $\mathsf{bad}_i \leftarrow (\overline{M}_j \neq \bot)$
Return $\bot$
```

Figure 16: Games $\mathrm{G}_2, \mathrm{G}_3$ for proof of Theorem 9.1, where the former includes the boxed code and the latter does not.

is distributed just like $(S_1, \overline{S}_1)$. Then again since $\alpha R \neq P_{w_2}$, we can apply the Lemma, with the randomness being β_2, γ_2, to conclude that (S_2, S_2') is distributed just like $(S_2, \overline{S}_2)$. Now consider the case that $y \in X$. Then αR equals exactly one of P_{w_1}, P_{w_2}. (Meaning, equals one of them but not the other. This uses that $P_{w_1} \neq P_{w_2}$, which is true due to the choices in Figure 15.) Line 17 sets bad_i if the undesired decryption succeeds. Now Lemma 9.2 again applies to say that (S_j, S_j') is distributed just like $(S_j, \overline{S}_j)$. So overall bad_i is set with the same probability as in game G_1. This concludes the justification of Equation (29).

Games $\mathrm{G}_2, \mathrm{G}_3$ are identical-until-bad, so by the Fundamental Lemma of Game Playing [8], we have

$$\begin{aligned}\Pr[\mathrm{G}_2(\mathcal{A})] &= \Pr[\mathrm{G}_3(\mathcal{A})] + (\Pr[\mathrm{G}_2(\mathcal{A})] - \Pr[\mathrm{G}_3(\mathcal{A})]) \\ &\leq \Pr[\mathrm{G}_3(\mathcal{A})] + \Pr[\mathrm{G}_2(\mathcal{A}) \text{ sets } \mathsf{bad}] \,. \end{aligned} \tag{30}$$

```
Game G_4

VOUCH(t):  // Adversary calls with a vector of triples
m ← |t|
Require: ID(t[1]), ..., ID(t[m]) are all distinct
For i = 1, ..., |t| do
    (y, id, ad) ← t[i] ; adct ←$ SE.Enc(adkey, ad)
    x ←$ 𝔽 \ XS; XS ← XS ∪ {x} ; z ← P_{adkey,a_1,...,a_t}(x) ; sh ← (x, z)
    S_1, S_2, S̄_1, S̄_2 ←$ 𝔾
    K_1 ← KDF(S_1) ; K_2 ← KDF(S_2) ; ct_1 ←$ SE.Enc(K_1, (adct, sh)) ; ct_2 ←$ SE.Enc(K_2, (adct, sh))
    K̄_1 ← KDF(S̄_1) ; K̄_2 ← KDF(S̄_2) ; M̄_1 ← SE.Dec(K̄_1, ct_1) ; M̄_2 ← SE.Dec(K̄_2, ct_2)
    bad_i ← ((M̄_1, M̄_2) ≠ (⊥, ⊥))
Return ⊥
```

```
Game G_5

VOUCH(t):  // Adversary calls with a vector of triples
m ← |t|
Require: ID(t[1]), ..., ID(t[m]) are all distinct
For i = 1, ..., |t| do
    (y, id, ad) ← t[i] ; adct ←$ SE.Enc(adkey, ad)
    x ←$ 𝔽 \ XS; XS ← XS ∪ {x} ; z ← P_{adkey,a_1,...,a_t}(x) ; sh ← (x, z)
    K_1 ←$ {0,1}^k ; K_2 ←$ {0,1}^k ; ct_1 ←$ SE.Enc(K_1, (adct, sh)) ; ct_2 ←$ SE.Enc(K_2, (adct, sh))
    K̄_1 ←$ {0,1}^k ; K̄_2 ←$ {0,1}^k ; M̄_1 ← SE.Dec(K̄_1, ct_1) ; M̄_2 ← SE.Dec(K̄_2, ct_2)
    bad_i ← ((M̄_1, M̄_2) ≠ (⊥, ⊥))
Return ⊥
```

Figure 17: Games $\mathrm{G}_4, \mathrm{G}_5$ for proof of Theorem 9.1.

We claim that

$$\Pr[\mathrm{G}_2(\mathcal{A}) \text{ sets } \mathsf{bad}] \leq \frac{2m}{q-1} \,. \tag{31}$$

To justify this, let $\beta = \alpha^{-1}$ be the inverse of α in the group $\mathbb{Z}_q^*$. The condition tested at line 13 of G_2 can equivalently be written as $R \in \{\beta\, P_{w_1}, \beta\, P_{w_2}\}$. Line 6 sets $R \leftarrow \textsc{Hash}(y)$. That $y \notin X$ (line 13) means that $\mathrm{HT}[y]$ was determined after $P_1, \ldots, P_{n'}$, and so the random choice of $\mathrm{HT}[y]$ has probability at most $2/(q-1)$ of being in the set $\{\beta\, P_{w_1}, \beta\, P_{w_2}\}$. (If y repeats in a triple at line 4, we can consider only the first time it occurs.) There are at most m choices of y so the union bound yields Equation (31).

We now need to bound $\Pr[\mathrm{G}_3(\mathcal{A})]$. With game G_4 as in Figure 17, we claim that

$$\Pr[\mathrm{G}_3(\mathcal{A})] \leq \Pr[\mathrm{G}_4(\mathcal{A})] \ . \tag{32}$$

We saw that $S_1, S_2, \overline{S}_1, \overline{S}_2$ in game G_3 were uniformly and independently distributed over $\mathbb{G}$. Game G_4 directly picks them as such. This renders various variables unused, allowing them to be dropped. In the $y \notin X$ case, game G_4 sets bad_i as in game G_3. In the $y \in X$ case, it sets bad_i more generously than in game G_3. This justifies Equation (32).

At lines 6,7, game G_5 in Figure 17 picks the keys $K_1, K_2, \overline{K}_1, \overline{K}_2$ directly at random from $\{0,1\}^k$ rather than, as in game G_4, obtaining them via KDF. We build adversary $\mathcal{A}_{\mathrm{kdf}}$ so that

$$\Pr[\mathrm{G}_4(\mathcal{A})] - \Pr[\mathrm{G}_5(\mathcal{A})] \leq \mathbf{Adv}^{\mathrm{kdf}}_{\mathsf{KDF},4m}(\mathcal{A}_{\mathrm{kdf}}) \ . \tag{33}$$

Adversary $\mathcal{A}_{\mathrm{kdf}}$ receives, from $\mathbf{G}^{\mathrm{kdf}}_{\mathsf{KDF},4m}.\textsc{Init}$, a list of $4m$ k-bit keys that we write as $K_{1,1}, K_{1,2}, \overline{K}_{1,1}, \overline{K}_{1,2}, \ldots, K_{m,1}, K_{m,2}, \overline{K}_{m,1}, \overline{K}_{m,2}$. Adversary $\mathcal{A}_{\mathrm{kdf}}$ starts by executing the steps of $\textsc{Init}$ in Figure 15. It then runs $\mathcal{A}$, itself replying to the latter's oracle queries. It replies to $\textsc{Post}$ and $\textsc{Hash}$ queries of $\mathcal{A}$ as per Figure 15. For the $\textsc{Vouch}$ query, it proceeds as in game G_5, executing lines 1–5. At lines 6,7, instead of picking $K_1, K_2, \overline{K}_1, \overline{K}_2$, it sets them to $K_{i,1}, K_{i,2}, \overline{K}_{i,1}, \overline{K}_{i,2}$, respectively. (Here i is the For loop counter from line 3.) It then reverts to following game G_5 for line 8, the response to the query being as per line 9. If the condition at line 12 of Figure 15 is met, our adversary $\mathcal{A}_{\mathrm{kdf}}$ returns 1 as its output guess bit, else 0. For the analysis, let $S_{1,1}, S_{1,2}, \overline{S}_{1,1}, \overline{S}_{1,2}, \ldots, S_{m,1}, S_{m,2}, \overline{S}_{m,1}, \overline{S}_{m,2}$ denote the seeds chosen at line 3 of $\mathbf{G}^{\mathrm{kdf}}_{\mathsf{KDF},4m}.\textsc{Init}$. Then $S_{i,1}, S_{i,2}, \overline{S}_{i,1}, \overline{S}_{i,2}$ play the role of $S_1, S_2, \overline{S}_1, \overline{S}_2$ in game G_4. So if the challenge bit d of game $\mathbf{G}^{\mathrm{kdf}}_{\mathsf{KDF},4m}$ is 1 then $\mathcal{A}$ gets the view it would in game G_4, and if d is 0 then $\mathcal{A}$ gets the view it would in game G_5, which implies Equation (33).

We can now appeal to the assumed robustness of SE to bound $\Pr[\mathrm{G}_5(\mathcal{A})]$. Namely, we build adversary $\mathcal{A}_{\mathrm{rob}}$ so that

$$\Pr[\mathrm{G}_5(\mathcal{A})] \leq \mathbf{Adv}^{\mathrm{rob\$}}_{\mathsf{SE}}(\mathcal{A}_{\mathrm{rob}}) \ . \tag{34}$$

Adversary $\mathcal{A}_{\mathrm{rob}}$ starts by executing the steps of $\textsc{Init}$ in Figure 15. It then runs $\mathcal{A}$, itself replying to the latter's oracle queries. It replies to $\textsc{Post}$ and $\textsc{Hash}$ queries of $\mathcal{A}$ as per Figure 15. For the $\textsc{Vouch}$ query, it proceeds as in game G_5, executing lines 1–5. At line 6, instead of picking K_1, K_2, ct_1, ct_2 as shown, it queries its own $\textsc{Enc}$ oracle to get $\perp \leftarrow\!\!\$\ \textsc{Enc}(2i-1, (adct, sh))$ and $\perp \leftarrow\!\!\$\ \textsc{Enc}(2i, (adct, sh))$. (Here i is the For loop counter from line 3.) It then queries its $\textsc{Dec}$ oracle to get $\perp \leftarrow \textsc{Dec}(2i-1, 2i-1)$ and $\perp \leftarrow \textsc{Dec}(2i, 2i)$. (These oracles always return $\perp$.) As per game G_5 it returns $\perp$ as the response to $\mathcal{A}$'s query. (It skips line 8, not setting bad_i.) When $\mathcal{A}$ halts, so does $\mathcal{A}_{\mathrm{rob}}$. For the analysis, let $K_1, K_2, \ldots, K_{2m-1}, K_{2m}$ and $\overline{K}_1, \overline{K}_2, \ldots, \overline{K}_{2m-1}, \overline{K}_{2m}$ denote the keys chosen at lines 1,2, respectively, of $\mathbf{G}^{\mathrm{rob\$}}_{\mathsf{SE},2m,2m}.\textsc{Init}$ of Figure 13. Then $K_{2i-1}, K_{2i}, \overline{K}_{2i-1}, \overline{K}_{2i}$ play the role of $K_1, K_2, \overline{K}_1, \overline{K}_2$ in game G_5. So if the execution of $\mathcal{A}$ with game G_5 sets bad_i for some i, then the execution of $\mathcal{A}_{\mathrm{rob}}$ with game $\mathbf{G}^{\mathrm{rob\$}}_{\mathsf{SE},2m,2m}$ sets win. This justifies Equation (34).

Putting together the above yields Equation (25). ▮

References

[1] Looseness, security risks, and LWR vs. LWE, June 2021. https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/Yx0wZuZP6ag.

[2] National Center for Missing and Exploited Children (NCMEC), June 2021. https://www.missingkids.org/theissues/csam.

[3] M. Abdalla, M. Bellare, and G. Neven. Robust encryption. In D. Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 480–497, Zurich, Switzerland, Feb. 9–11, 2010. Springer, Heidelberg, Germany.

[4] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545, Kyoto, Japan, Dec. 3–7, 2000. Springer, Heidelberg, Germany.

[5] M. Bellare, C. Namprempre, and G. Neven. Unrestricted aggregate signatures. In L. Arge, C. Cachin, T. Jurdzinski, and A. Tarlecki, editors, *ICALP 2007*, volume 4596 of *LNCS*, pages 411–422, Wroclaw, Poland, July 9–13, 2007. Springer, Heidelberg, Germany.

[6] M. Bellare, R. Ng, and B. Tackmann. Nonces are noticed: AEAD revisited. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 235–265, Santa Barbara, CA, USA, Aug. 18–22, 2019. Springer, Heidelberg, Germany.

[7] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *ACM CCS 93*, pages 62–73, Fairfax, Virginia, USA, Nov. 3–5, 1993. ACM Press.

[8] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

[9] M. Bellare and B. Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 247–276, Santa Barbara, CA, USA, Aug. 14–18, 2016. Springer, Heidelberg, Germany.

[10] D. Bernstein. On the looseness of FO derandomization. Cryptology ePrint Archive, Report 2021/912, 2021. https://eprint.iacr.org/2021/912.

[11] A. Bhowmick, D. Boneh, S. Myers, K. Talwar, and K. Tarbe. The Apple PSI system, July 2021.

[12] P. Bose, V. T. Hoang, and S. Tessaro. Revisiting AES-GCM-SIV: Multi-user security, faster key derivation, and better bounds. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 468–499, Tel Aviv, Israel, Apr. 29 – May 3, 2018. Springer, Heidelberg, Germany.

[13] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145, Las Vegas, NV, USA, Oct. 14–17, 2001. IEEE Computer Society Press.

[14] P. Farshim, B. Libert, K. G. Paterson, and E. A. Quaglia. Robust encryption, revisited. In K. Kurosawa and G. Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 352–368, Nara, Japan, Feb. 26 – Mar. 1, 2013. Springer, Heidelberg, Germany.

[15] P. Farshim, C. Orlandi, and R. Roşie. Security of symmetric primitives under incorrect usage of keys. *IACR Trans. Symm. Cryptol.*, 2017(1):449–473, 2017.

[16] P. Grubbs, J. Lu, and T. Ristenpart. Message franking via committing authenticated encryption. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 66–97, Santa Barbara, CA, USA, Aug. 20–24, 2017. Springer, Heidelberg, Germany.

[17] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467, Miami Beach, Florida, Oct. 19–22, 1997. IEEE Computer Society Press.

[18] P. Rogaway. Authenticated-encryption with associated-data. In V. Atluri, editor, *ACM CCS 2002*, pages 98–107, Washington, DC, USA, Nov. 18–22, 2002. ACM Press.

A Authenticity tightly implies ROB$

Correctness of the protocol as per Theorem 9.1 relied on the ROB$ security of the symmetric encryption scheme SE. The SE Apple is using, however, is an authenticated encryption scheme. The definition of the latter does not explicitly require robustness. The gap is bridged by the fact, noted in [11], that authenticity implies the type of robustness needed here.

In our concrete-security setting, however, we need to also ask what is the tightness of the reduction and whether it holds in our multi-user setting. Lemma A.1 below confirms that multi-user authenticity tightly implies ROB$.

The definition of authenticity we use, called AUTH, starts from the INT-CTXT definition of [4], extended to the multi-user setting as per [9], and then weakens that in two ways. First, the $\textsc{Enc}$ oracle, that would normally allow the adversary to obtain encryptions of messages of its choice for users of its choice, is omitted. Second, rather than returning the result of decryption to the adversary, the decryption oracle $\textsc{Dec}$ returns $\perp$. Lemma A.1 says that even this weak notion of authenticity tightly implies ROB$.

```
Game $\mathbf{G}^{\mathrm{auth}}_{\mathsf{SE},r}$

INIT:
1 For $i = j, \ldots, r$ do $\overline{K}_j \leftarrow\!\!\$\ \{0,1\}^k$

DEC$(j, C)$:  // Adversary calls with user $j \in [1..r]$ and ciphertext $C$
2 Require: $1 \leq j \leq r$
3 $M \leftarrow \mathsf{SE.Dec}(\overline{K}_j, C)$
4 If $(M \neq \bot)$ then $\mathsf{win} \leftarrow \mathsf{true}$
5 Return $\bot$

FIN:  // No adversary input
6 Return $\mathsf{win}$
```

Figure 18: Game defining AUTH security of SE.

Proceeding to the details, the definition is based on game $\mathbf{G}^{\mathrm{auth}}_{\mathsf{SE},r}$ of Figure 18. If $\mathcal{A}_{\mathrm{auth}}$ is an adversary playing this game, its advantage is

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathsf{SE},r}(\mathcal{A}_{\mathrm{auth}}) = \Pr[\mathbf{G}^{\mathrm{auth}}_{\mathsf{SE},r}(\mathcal{A}_{\mathrm{auth}})] \ . \tag{35}$$

Integer $r \geq 1$ is the number of users. Line 1 picks keys for all of them. The adversary can call oracle DEC with a user identity j and a ciphertext C of its choice. Oracle DEC decrypts C under the key $\overline{K}_j$ of receiver j. The adversary wins if this decryption is non-$\bot$.

Lemma A.1 Let SE be a symmetric encryption scheme, and $\mathcal{A}_{\mathrm{rob}}$ an adversary playing game $\mathbf{G}^{\mathrm{rob\$}}_{\mathsf{SE},s,r}$. Assume it makes q_d queries per user to its DEC oracle. Then we can construct an adversary $\mathcal{A}_{\mathrm{auth}}$ such that

$$\mathbf{Adv}^{\mathrm{rob\$}}_{\mathsf{SE},s,r}(\mathcal{A}_{\mathrm{rob}}) \leq \mathbf{Adv}^{\mathrm{auth}}_{\mathsf{SE},r}(\mathcal{A}_{\mathrm{auth}}) \ . \tag{36}$$

Adversary $\mathcal{A}_{\mathrm{auth}}$ makes q_d queries per user to its DEC oracle. Its running time is about that of $\mathcal{A}_{\mathrm{rob}}$.

Proof of Lemma A.1: Adversary $\mathcal{A}_{\mathrm{auth}}$ begins by picking keys $K_1, \ldots, K_s \leftarrow\!\!\$\ \{0,1\}^k$, where $\{0,1\}^k$ is the key space of SE, as per line 1 of game $\mathbf{G}^{\mathrm{rob\$}}_{\mathsf{SE},s,r}$. It initializes counter w to 0. It then runs $\mathcal{A}_{\mathrm{rob}}$. When $\mathcal{A}_{\mathrm{rob}}$ makes a query i, M to its ENC oracle, adversary $\mathcal{A}_{\mathrm{auth}}$ executes $w \leftarrow w + 1$ and $C_w \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_i, M)$, as per lines 4,5 of game $\mathbf{G}^{\mathrm{rob\$}}_{\mathsf{SE},s,r}$. It returns $\bot$ to $\mathcal{A}_{\mathrm{rob}}$. When $\mathcal{A}_{\mathrm{rob}}$ makes a query j, l to its DEC oracle, adversary $\mathcal{A}_{\mathrm{auth}}$ calls its own DEC oracle with j, C_l. It returns $\bot$ to $\mathcal{A}_{\mathrm{rob}}$. ▮