LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Dissertation

an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

Formal Verification of Revocation Approaches in Identity-Based Cryptography

vorgelegt von
Sophia Elisabeth Grundner-Culemann
am 15. September 2024

LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Dissertation

an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

Formal Verification of Revocation Approaches in Identity-Based Cryptography

Sophia Elisabeth Grundner-Culemann

Erster Gutachter: Prof. Dr. Dieter Kranzlmüller
Zweiter Gutachter: Prof. Dr. Wolfgang Hommel
Tag der mündlichen Prüfung: 20. Dezember 2024

Eidesstattliche Versicherung

Hiermit versichere ich gemäß Promotionsordnung vom 12.07.11, §8, Abs. 2, Pkt. 5 eidesstattlich, dass ich die Dissertation selbstständig und ohne unerlaubte Beihilfe angefertigt habe. Bei der sprachlichen Abfassung habe ich ebenfalls keine Hilfe in Anspruch genommen. (Erklärung gemäß Promotionsordnung vom 12.07.11, §8, Abs. 2, Pkt. 6)

München, den 13.02.2025

Sophia Grundner-Culemann

. . . . . .

Unterschrift

Danksagung

A research mindset is a curiosity about the information that isn't being offered.
– Mike Rosulek

An dieser Stelle möchte ich mich herzlich bei allen bedanken, die mich bei dieser Arbeit und den verschiedenen Projekten in meiner Promotionszeit unterstützt haben.

Zuvorderst gilt mein herzlicher Dank meinem Doktorvater, Herrn Prof. Dr. Dieter Kranzlmüller, für seinen steten Rückhalt, sein immer offenes Ohr und die vielen Gelegenheiten, die er mir bot, mich in jederlei Hinsicht auszuprobieren.

Herrn Prof. Dr. Wolfgang Hommel danke ich für die sorgfältige Zweitbegutachtung meiner Arbeit - und dafür, dass er im WS 15/16 zusammen mit Prof. Reiser die Vorlesung zu IT-Sicherheit abgehalten und mich so in dieses Forschungsgebiet eingeführt hat.

Ich danke auch Herrn Prof. Dr. Jasmin Blanchette, der freundlicherweise den Prüfungsvorsitz für mein Rigorosum übernommen hat, und Herrn Prof. Dr. Johannes Kinder, der als Ersatzprüfer bereitstand.

Für sehr liebenswürdige und prompte Hilfe gebührt auch Dr. Ralf Sasse und seinem Team an der ETH Zürich, davon insbesondere Xenia, Sofia und Felix, herzlicher Dank: Sie haben mir geholfen, meiner Beweisidee nachhaltig Leben einzuhauchen. Diese Tage bei euch waren für mein Vorankommen spielentscheidend. Danke, dass ihr für mich in Zürich einen Schreibtisch und viel Zeit hattet!

Mein nächster Dank gebührt dem MNM-Team, dem ich so gern angehöre:

Vitalian, danke für "Schreib einfach mal Grütze!" - das war wie ein Zauberwort. Karl, Annette, Miki, Sigrid, danke, dass ihr den Laden auf verschiedene Weise am Laufen haltet. Dadurch konnte ich erstens mehr mit Lehre und Forschung als mit Papierkram und Papierstau beschäftigen und hatte zweitens immer einen Vertrag, einen funktionierenden Laptop und reibungslose Dienstreisen.

Meinen Doktorgeschwistern verschiedener "Generationen" - Jan, Tobi G., Tobi F., Tobi W., Pascal, Roger, Markus ("Plan nicht mehr als du arbeitest!"), Matthias, Cuong, Amir, Max, Nils R. und Klemens, Nils M., Stefan-Lukas, Minh, Dang, Korbinian, Michelle, Sergej, Fabian, Daniel D., Flo, Fabio, David, Daniel K., Miran, Michael - vielen Dank für alles, von Paper-Gegenlesen vor knappen Deadlines über "Isogenius"-Mini-Jourfixes bis hin zu Klausurkorrekturen, Seminar-Wohl und -Wehen, Probevorträge, diskussionsreichen "richtigen" Jour Fixes, vietnamesischen Süß- und Salzspeisen, Spaziergängen im Englischen Garten bei jedem Wetter, Badehosen auf Gang E, "Isoweek", "Kuh, Bits & Berge", Bichteln & Wallern, Wandertagen, Christbaumloben und jede Menge Rat & Tat zu jeder Tageszeit. Ihr seid die Besten. Dank euch bin ich jeden Tag wieder gerne ins Büro gekommen.

Nils, danke für deine riesengroße Unterstützung über all die Jahre - von meiner Masterarbeit über meine ersten Jahre am Lehrstuhl bis zum letzten Teil des Marathons, für den du mein akademischer und persönlicher Cheerleader warst.

Tobi - ich danke dir für alles, was ich von dir lernen durfte, und die vielen schönen und manchmal schwierigen Zeiten, die wir verbracht haben. Du hast mich sofort unter deine Fit-

tiche genommen, mir Paper-Schreiben, Fragenstellen, wissenschaftlich Denken beigebracht. Ohne dich hätte ich nie eine Quantencomputing- oder Krypto-Vorlesung gehalten und mich auch viele andere Dinge nicht getraut. Dank dir weiß ich genau, wie glücklich Paper-Nachtschichten im Büro machen können und dass der Weg nach Kirchheim im Flug vergehen kann. Unsere Spaziergänge haben in alle Ecken des südlichen Englischen Gartens geführt und unsere Gespräche in alle Winkel des Lebens. Danke für all deine Zeit, Verbundenheit und Freundschaft.

Auch einige Team-Mitglieder und gute Feen am Leibniz-Rechenzentrum und der Universität der Bundeswehr seien genannt: Helga (besonders für Raum-Orga und Corona-Test-Kontroll-Schichten), Herr Prof. Dr. Reiser (für die eingangs erwähnte Vorlesung und guten Rat, besonders in meinen ersten Monaten), Sabrina, Susanne, Severine, Peter, Florent, Lizzy, Corinna, Andre, Josef, Florian: Dank euch waren Veranstaltungen möglich, Projekte gelungen, Studierende gut versorgt und die Stimmung bestens.

Noch vielen anderen ist zu danken, die das Informatikinstitut und meine lange Zeit dort lebenswert gemacht haben: Dr. Jan Johannsen, Steffen, Prof. Bry, Andrea und Anna und ihren Kolleg:innen vom DBS-Lehrstuhl, Maximilian, Alex, Basti, Alexis, Uli, meinen vielen engagierten Studierenden, dem Mensa-Team, dem Reinigungs-Team, dem Team an der und um die Pforte (allen voran Ehsan für viel Tee und viele liebe Worte). Frau Prof. Dr. Caroline Friedel danke ich für ihre Unterstützung im Mentoring-Programm.

Meine Promotionszeit wäre nicht halb so schön gewesen ohne mein liebes QuaSiModO-Team, allen voran Daniel, Stefan-Lukas, Mörg und Alexander. Danke euch für so viel schöne Zeit. Eure Freundschaft möchte ich nicht missen. Darüber hinaus habe ich im genu-verse viele weitere großartige Menschen kennengelernt. Mein Leben wäre ohne geek weeks nicht das gleiche (und nirgends schreibt es sich so gut an der Diss wie auf der session).

Danke auch an Andreas, Niko, Sven, Wolfgang und Sabine, dass wir zusammen die Praktika organisiert (und im März 2020 spontan remote-isiert) und den Quantenzirkel ins Leben gerufen haben. Vielleicht gehen wir ja mal wieder auf ein “Quantensushi”.

Danken möchte ich außerdem dem crypt@b-it-Team Michael, Marcel, Johanna, Alex und Janik - toll, dass ich mitmachen und später mit-organisieren durfte! Auf meiner ersten crypt@b-it hat mich besonders Mike Rosuleks ruhiger und verständlicher Vorlesungsstil beeindruckt. Seine Webseite “So You’re Starting a PhD?” kann ich nur jede:m ans Herz legen - auch für andere Projekte als Doktorarbeiten. Thank you, Mike!

Zuletzt, aber umso ausdrücklicher, möchte ich meiner Familie und meinen Freund:innen danken. Ihr habt mich, jede:r auf die eigene Art, während dieser Jahre mit Rat und Tat begleitet. Eure Unterstützung war mehr als ein Absatz erzählen kann, und dafür schätze ich mich immens glücklich.

Abstract

Traditional public key cryptography relies on certificates to attribute a public key to the owner. This approach produces significant cost in secure communication because certificates need to be downloaded and their validity status needs to be tracked. Identity-based Cryptography (IBC) offers an alternative by deriving public keys directly from their owners' unique identifiers, e.g. email addresses. This method eliminates the need to manage certificates. A Trusted Third Party (TTP) generates the user's corresponding secret keys.

A problem arises when a key is revoked because the owner must be able to continue using their identifier. There are several proposals to handle the revocation problem in IBC. However, there is little abstract analysis of this problem and the security considerations are limited to manual proofs for specific approaches and informal explanations of general properties. This work gives a broader overview: It systematizes revocation mechanisms overall, identifies three classes of revocation mechanisms in IBC and provides a formalization for an automated prover to analyze the security properties expected of each class.

The overall systematization happens along two dimensions: 1. "explicit/implicit" (i.e. validity either needs to be checked or can be ignored by third parties) and 2. "directly/indirectly" (i.e. the mechanism manages revoked keys or it manages valid keys).

All identity-based methods are implicit and indirect, obsoleting all secret keys at a certain cue and issuing new material to all non-revoked users. They differ in how users obtain new key material.

This work identifies: 1. The renewal method, where keys are completely replaced, 2. the individual-update method, where each user receives a customized update token for their individual key, and 3. the universal-update method, where all users receive the same update token for their individual keys.

The formalization captures the stages that all classes have in common and the process-agnostic security goals they ideally achieve. It is adaptable to each class through the mathematical dependencies it models for the keys and yields a blueprint model for an automated analysis with a trace-based prover tool. The proof-of-concept implementation in Tamarin is the first high-level security analysis of IBC-revocation mechanisms. It confirms both the weaknesses to decryption key exposure that were found in individual update mechanisms if they fail to re-randomize the key in the update process and the collusion attack for the universal update approach that was previously only acknowledged indirectly. It also reconciles the notions of Decryption Key Exposure Resistance and of Forward-/Post-Compromise Security, which were previously never discussed together.

Based on this work, research might consider more detailed mathematical context of a certain IBC scheme and its revocation mechanism if and when the problem of modeling distributive laws is solved. This would allow for a more fine-grained security analysis and increase trust in revocable identity-based mechanisms. More generally, the proposed formalization can be used as blueprint for the formal verification of obsolescence-based revocation mechanisms and applications that use key updates outside of IBC.

Zusammenfassung

Traditionelle Public-Key-Kryptographie nutzt Zertifikate, um öffentliche Schlüssel ihren Eigentümerinnen zuzuordnen. Dieser Ansatz ist teuer, da man die Zertifikate herunterladen und ihre Gültigkeit nachverfolgen muss. Identitäts-basierte Kryptographie (IBK) erlaubt es, öffentliche Schlüssel direkt aus eindeutigen Identifikatoren, wie z.B. E-Mail-Adressen, abzuleiten. So erübrigt sich die Zertifikatsverwaltung. Eine vertrauenswürdige dritte Partei (TTP) erzeugt die entsprechenden geheimen Schlüssel der Nutzerinnen.

Ein Problem tritt auf, wenn ein Schlüssel widerrufen wird, denn die Eigentümerin soll ihren Identifikator behalten. Trotz mehrerer konkreter Schlüsselwiderruf-Ansätze in IBK gibt es nur wenig abstrakte Analyse. Deren Sicherheitsbetrachtungen beschränken sich jedoch auf manuelle Beweise für bestimmte Ansätze oder sind informell. Diese Arbeit bietet einen umfassenderen Blick: Sie systematisiert Widerrufmechanismen im Allgemeinen, identifiziert drei Klassen von Widerrufmechanismen in IBK und liefert eine Formalisierung zur Analyse von Sicherheitseigenschaften jeder Klasse mithilfe eines automatisierten Beweisers.

Die allgemeine Systematisierung hat zwei Dimensionen: 1. „explizit/implizit“ (d.h. die Gültigkeit muss von Dritten entweder überprüft werden oder kann ignoriert werden) und 2. „direkt/indirekt“ (d.h. der Mechanismus verwaltet widerrufene Schlüssel oder verwaltet gültige Schlüssel). Alle identitäts-basierten Methoden sind implizit und indirekt: Alle geheimen Schlüssel verlieren zu einem bestimmten Zeitpunkt ihr Gültigkeit und alle nicht-widerrufenen Benutzerinnen erhalten neue Schlüssel.

Die Methoden unterscheiden sich in der Generierung neuer Schlüssel. Diese Arbeit identifiziert: 1. Re-keying, wobei der Schlüssel vollständig ersetzt wird, 2. individuelle Aktualisierungstoken, die jede Benutzerin für ihren individuellen Schlüssel erhält, 3. universelle Aktualisierungstoken, die alle Benutzerinnen verwenden können.

Die Formalisierung beschreibt die Widerruf-Phasen und die Sicherheitsziele, die alle drei Klassen idealerweise erreichen. Sie ist für jede Klasse durch die mathematischen Abhängigkeiten, die sie für die Schlüssel modelliert, anpassbar und liefert eine Blaupause zur Analyse mit einem Trace-basierten automatisierten Beweiser. Die Proof-of-Concept-Implementierung in Tamarin ist die erste abstrakte Sicherheitsanalyse von IBK-Widerrufsmechanismen. Sie bestätigt sowohl die Schwächen gegenüber Decryption-Key Exposure, die in Mechanismen für individuelle Aktualisierungstoken gefunden wurden, wenn die Schlüssel bei der Aktualisierung nicht re-randomisiert werden, als auch den Kollusionsangriff bei der universellen Aktualisierungsmethode, der zuvor nur indirekt anerkannt wurde. Außerdem wird Decryption-Key Exposure-Resistenz mit Forward-/Post-Compromise-Sicherheit in Einklang gebracht; die drei Begriffe wurden zuvor nie gemeinsam betrachtet.

Basierend auf dieser Arbeit könnten detailliertere mathematische Zusammenhänge von IBK-Widerruf-Schemata betrachtet werden, falls das Problem der Modellierung distributiver Gesetze in automatisierten Beweisern gelöst wird. Dies würde detailliertere Sicherheitsanalysen ermöglichen und das Vertrauen in widerrufbare identitäts-basierte Mechanismen stärken. Außerdem ist die vorgeschlagene Formalisierung eine Blaupause für die formale Verifizierung von obsoleszenzbasierten Widerrufmechanismen und Anwendungen jenseits von IBK.

Contents

1. Introduction

Asymmetric cryptography is important for many digital cryptographic processes such as authentication or key agreement. "Asymmetric" cryptography means that there are two keys in every interaction: One secret key (with which a user can digitally sign a message, for example) and one public key (with which another user can verify the signature, but can not sign another message themself).

In the most common key management system, the Public Key Infrastructure (PKI), the connection between a user and their key is asserted by a commonly trusted Certificate Authority (CA): Users register their public key with the CA and provide credible evidence of both their identity and their ownership of the key they present. If the CA is sure that the user has provided their true identity and legitimately holds the secret key that corresponds to the presented public key, it issues a corresponding public key certificate.

This process is tedious because every communication partner who wants to use a public key needs to check the corresponding certificate first. When time or resources are scarce, these certificates can become a bottleneck. An example is the Internet of Things (IoT), where many devices have very limited computing power, small storage, or use a small battery [Gug20].

Adi Shamir [Sha85] proposed a different handling of public keys, namely Identity-based Cryptography (IBC). In this approach, the public key is inherently bound to the identity of its owner by the underlying mathematics of the cryptographic scheme. A person's identity can, for example, be their e-mail-address. When encrypting an e-mail to a user Alice, for example, one can simply derive her public key from her address "alice@ibc.edu" (which must be known anyway to communicate with her). Thus, there is no overhead for looking up and checking her public key certificate.

In a typical PKI, the secret is chosen freely and the public key is computed accordingly. In IBC, the process is reversed: The public key (= the identity) is fixed, so the secret key needs to be derived from it. A user must obviously not be able to determine their own secret key: Otherwise, any user could calculate any other user's private key from their identity. Therefore, the computation of the secret key is outsourced to a so-called Trusted Third Party (TTP), who is assumed to be trustworthy and to have a secure channel with all users. The TTP has its own key material, namely a Master Public Key (MPK) and a Master Secret Key (msk). It uses the msk to derive the users' secret keys. For any cryptographic operation that requires a user's public key, the master public key is also necessary: This ensures the mathematical connection between the user's identity, the user's secret key, and the TTP, so that the system works and is trustworthy.

The difference in key management is also consequential for the key revocation, since the revocation mechanisms used in typical PKI are inconsistent with the IBC ideal of reducing management overhead. PKI revocation relies on Certificate Revocation Lists (CRL), where revoked keys are blacklisted, or the certificates expire; other mechanisms rely on whitelisting valid keys. However, in IBC, the core idea is that users can simply derive a public key from another user's identity without checking the keys validity through third-party documents, so black- or whitelisting would undermine the main idea.

1.1. Research Questions

There are various proposals for key revocation in IBC, but the corresponding papers either only graze the topic or propose specific schemes which are evaluated along varying security notions. The literature lacks a systematic treatment of the topic to gain general insights in the revocation problem for IBC.

Furthermore, specific IBC-revocation schemes and their security properties are mostly analyzed only manually (and sometimes corrected, see for example [SE13b]). Computer-aided analyses improve the understanding of security in many areas [Tri20; Sch+12; Coh+20; BDS17]; for the cryptographic level (rather than, for example, the implementation level), trace-based tools like Tamarin or ProVerif are popular tools. Computer-aided analyses of IBC exist ([BHS19; AB22]), but none specifically covers revocation[1].

We fill this gap. While the findings from related work mainly point to the limitations of analyzing IBC with automated provers, our work suggests that analyzing IBC-revocation is still worthwhile. This is because insights about revocation can be gained on a protocol level, whereas analyzing specific schemes requires more detailed mathematical modeling.

The main research questions answered in this work is the following:

> *How can formal verification with an automated prover be used to assess and confirm the security properties of various revocation techniques in Identity-Based Cryptography?*

It is answered by considering the following sub-questions:

RQ 1) What, if any, are the systematic commonalities and differences between various revocation approaches in IBC?

RQ 2) How can IBC-revocation approaches and their security properties be formally modeled such that a trace-based automated prover can reason about them?

RQ 3) How can the formalization be translated for an automated prover?

RQ 4) Does the formal verification confirm the assumed security properties?

Through the systematic analysis of commonalities and differences among various revocation approaches in IBC (**RQ 1)**), we understand the characteristics that a formal model must reflect. Answering **RQ 2)**, we understand how to account for differences between the approaches while retaining a unified model of their security on the level an automated prover addresses. Translating these models into a format suitable for an automated prover pursuant to **RQ 3)** ensures that the formalization is implementable and shows its limits in modeling the approaches. By running the models and evaluating the results, we answer **RQ 4)** and gain a deeper understanding of IBC-revocation security.

1.2. Methodology

We answer the research questions by conducting a literature review to identify and compare various IBC-revocation approaches. The literature is selected according to its impact and the

[1] There is one publication that aims to cover both, but the results are doubtful [SVR21]; see Section 3.2.1.

novelty of the described approach compared to previous work. The main aspects highlighted about the described approaches are collected and assembled in a comprehensive overview.

The commonalities identified between the approaches allow us to postulate a systematization of key revocation approaches in general. The systematic differences identified in the comparison yield a classification into three general approaches. The literature review further exhibits five security notions suitable for analysis in an automated prover.

The formalization of IBC-revocation is constructed by addressing the building blocks of an automated prover model. Using a minimal example for an IBC-revocation process, we identify the possible states and interactions of the agents in a communication scenario to be reflected in a formal verification model. The security properties are considered in more detail and adapted to account for the similarity between certain notions.

Tangible results are obtained by translating the formalization to four distinct formal verification models. We discuss the drawbacks and strengths found in the analysis, compare the results to previous findings, reconcile differences between the different security notions considered in literature, and point out the limitations of the analysis.

1.3. Contributions

- We systematize revocation mechanisms in general and place IBC within it.
- We propose a classification of existing revocation mechanisms in IBC.
- We formalize IBC-revocation mechanisms and the corresponding security requirements in a format that is suitable for automated reasoning.
- We provide a proof-of-concept implementation of the formalization and a blueprint for further implementations.
- We reconcile the security notion of Decryption Key Exposure Resistance with Forward-/Post-Compromise Security and review the claimed security properties in the considered schemes with automated analysis.

1.4. Structure of this work

Beyond the background explained in Chapter 2, this work is structured as follow:

- Chapter 3 systematically describes the main literature on IBC-revocation, on IBC-verification, and on automated analysis of other revocation approaches.
- Chapter 4 discusses similarities and differences of various IBC-revocation techniques found in related work. We identify three abstract classes.
- Chapter 5 unifies the three classes in a formalization that reflects the main processes in IBC-revocation. It serves as a basis for the specific analysis of each class with respect to the security notions derived from the literature.
- To show that the formalization is useful and suitable for automated reasoning, it is translated to a blueprint for automated analysis. Using this, four Tamarin models are implemented as a working example in Chapter 6.

- Chapter 7 evaluates and discusses the results of the automated analysis, both in their own right and compared to previous manual analyses.

1.5. Publications

This section lists the publications by the author, first those related to the thesis topic, then further publications that are not related to the thesis topic.

Publications that are related to this work

1. Nils gentschen Felde, Sophia Grundner-Culemann, and Tobias Guggemos. "Authentication in dynamic groups using identity-based signatures". In: *14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. Limassol, Cyprus, 2018. DOI: 10.1109/WiMOB.2018.8589148

   Summary: [2] This publication presents the idea of using identity-based schemes in group communication for sender authentication and introduces the mathematically verifiable revocation of the keys by re-calculating all keys in the system (including master key material). It also presents a taxonomy for choosing IBS-schemes for constrained scenarios.

   Own contribution: S. Grundner-Culemann contributed the taxonomy for IBS schemes, selected the schemes to evaluate and validated the re-key mechanism. All authors wrote and edited the paper together.

   Other contributions: N. gentschen Felde contributed the description of the scenario and its requirements for reliable access management. T. Guggemos introduced a Re-Key phase to the selected schemes and contributed the implementation and evaluation of the chosen schemes as well as the integration of IBS into a group key management architecture and the distribution of the keys with a Group Key Management Protocol.

   Relevance for this work: The proposed revocation method is mentioned as related work outside the scope (see final paragraph of Section 3.1).

2. Sophia Grundner-Culemann and Dieter Kranzlmüller. "EUF-ID-UPD-CMA: A security notion for key-updatable identity-based signature schemes". In: *crypto day matters 32*. Ed. by Stefan-Lukas Gazdag, Daniel Loebenberger, and Michael Nüsken. Bonn: Gesellschaft für Informatik e.V. / FG KRYPTO, 2021. DOI: 10.18420/cdm-2021-32-34

   Summary: A new notion for "EUF-ID-UPD-CMA"-security (Existential unforgeable under chosen message, identity and update-period) could help understand revocation mechanisms for identity-based signature keys, specifically the security of universal updates. This publication proposes a research outline with which a meaningful notion could be derived and proved useful.

   Own contribution: S. Grundner-Culemann proposed the security notion and wrote this research outline.

[2] To avoid discrepancies, this summary and the descriptions of the contributions quote the same section in [Gug20] almost verbatim.

Relevance for this work: The publication is an abstract for a presentation the author held at the 32nd "Krypto-Tag" organized by the *Gesellschaft für Informatik*. The presentation included an overview of related work, part of which was compared more extensively in this work ([BGK08; Gug20], see Table 4.1).

3. Sophia Grundner-Culemann. "A Survey of Revocation Mechanisms in Identity-based Cryptography". In: *crypto day matters 34*. Ed. by Daniel Loebenberger and Michael Nüsken. Bonn: Gesellschaft für Informatik e.V. / FG KRYPTO, 2022. DOI: 10.18420/cdm-2022-34-03

   Summary: This publication introduces the idea that there are four different flavors of key revocation (expiration, blacklisting, whitelisting, and obsolescence) and that identity-based approaches use either expiration or obsolescence, and reviews important work in this area together with a list of characteristics that distinguishes revocation mechanisms.

   Own contribution: S. Grundner-Culemann proposed the four revocation classes, discussed the classification of IBS, conducted the literature review and wrote the text.

   Relevance for this work: The publication is an abstract for a presentation that S. Grundner-Culemann held at the 34th "Krypto-Tag" organized by the *Gesellschaft für Informatik*. The presentation contained a first proposal to think about revocation along two dimensions, and the following discussions inspired changes that finally led to the version presented in Section 4.2. Further, the talk contained a comparison of various revocation approaches similar to Table 4.1 (with fewer related work items and partly different characteristics).

4. Tobias Guggemos and Sophia Grundner-Culemann. "group Identity Based Signatures: Efficiently revoking signing keys in communication groups". In: *crypto day matters 32*. Ed. by Stefan-Lukas Gazdag, Daniel Loebenberger, and Michael Nüsken. Bonn: Gesellschaft für Informatik e.V. / FG KRYPTO, 2021. DOI: 10.18420/cdm-2021-32-33

   Summary: This publication explains, recalling own previous research, the use of symmetric tokens for key revocation in identity-based signatures and the examples (two pairing-based and two Schnorr-like signature schemes that are adapted accordingly). The publication is an abstract for a presentation.

   Own contribution: For this mechanism, S. Grundner-Culemann contributed correctness checks for the integration of the update token in existing schemes, and helped choose the schemes to which the transformations are applied. Therefore, she helped review and edit the first draft of this publication.

   Other contributions: In previous work, T. Guggemos developed the underlying idea, applied it to selected schemes and discussed performance and security. He wrote the first draft of this presentation abstract and presented it at the 34th "Krypto-Tag" organized by the *Gesellschaft für Informatik*.

   Relevance for this work: The mechanism recalled in this publication [Gug20] is the only representative of the *universal update* class formally analyzed in this work. In this work, it is first introduced in the final paragraph of Section 3.1 and repeatedly referenced after that.

Other publications

1. Joo Cho, Stefan-Lukas Gazdag, Alexander von Gernler, Helmut Grießer, Sophia Grundner-Culemann, Tobias Guggemos, Tobias Heider, and Daniel Loebenberger. "Towards Quantum-resistant Virtual Private Networks". In: *crypto day matters 31*. Ed. by Marcel Selhorst, Daniel Loebenberger, and Michael Nüsken. Bonn: Gesellschaft für Informatik e.V. / FG KRYPTO, 2019. DOI: 10.18420/cdm-2019-31-22

2. Stefan-Lukas Gazdag, Sophia Grundner-Culemann, Tobias Guggemos, Tobias Heider, and Daniel Loebenberger. "A formal analysis of IKEv2's post-quantum extension". In: *Annual Computer Security Applications Conference*. ACM Digital Library. New York, NY, United States: Association for Computing Machinery, 2021, pp. 91–105. ISBN: 9781450385794. DOI: 10.1145/3485832.3485885

3. Stefan-Lukas Gazdag, Sophia Grundner-Culemann, Tobias Guggemos, Tobias Heider, and Daniel Loebenberger. "Migration zu quantenresistenter IT". in: *Sicherheit in vernetzten Systemen: 28. DFN-Konferenz*. Ed. by Albrecht Ude. Books on Demand, 2021, E-1–E-23. ISBN: 9783753448848

4. Stefan-Lukas Gazdag, Sophia Grundner-Culemann, Tobias Guggemos, Tobias Heider, and Daniel Loebenberger. "Entangled Secrets". In: *Linux Magazine* 247 (2021), pp. 16–19. URL: https://www.linux-magazine.com/Issues/2021/247/Quantum-Computing-and-Encryption (visited on 07/02/2021)

5. Stefan-Lukas Gazdag, Sophia Grundner-Culemann, Tobias Guggemos, Tobias Heider, and Daniel Loebenberger. "Migration zu quantenresistenter IT". in: *Linux Magazin* 04 (2021), pp. 84–88. URL: https://www.linux-magazin.de/ausgaben/2021/04/kryptographie/ (visited on 07/02/2021)

6. Stefan-Lukas Gazdag, Sophia Grundner-Culemann, Tobias Guggemos, Tobias Heider, and Daniel Loebenberger. "Quantensichere IT: ein Blick in die Glaskugel". In: *DFN Mitteilungen* 99 (2021), pp. 34–38. URL: https://www.dfn.de/fileadmin/5Presse/DFNMitteilungen/DFN_Mitteilungen_99.pdf (visited on 07/02/2021)

7. Stefan-Lukas Gazdag, Sophia Grundner-Culemann, Tobias Heider, Daniel Herzinger, Felix Schärtl, Joo Yeon Cho, Tobias Guggemos, and Daniel Loebenberger. "Quantum-Resistant MACsec and IPsec for Virtual Private Networks". In: *Security Standardisation Research*. Ed. by Felix Günther and Julia Hesse. Cham: Springer Nature Switzerland, 2023, pp. 1–21. ISBN: 978-3-031-30731-7. DOI: 10.1007/978-3-031-30731-7_1

8. Nils Mäurer and Sophia Grundner-Culemann. "Formal Verification of the LDACS MAKE Protocol". In: *crypto day matters 34*. Ed. by Daniel Loebenberger and Michael Nüsken. Bonn: Gesellschaft für Informatik e.V. / FG KRYPTO, 2022. DOI: 10.18420/cdm-2022-34-24

9. Nils Mäurer, Tobias Guggemos, Thomas Ewert, Thomas Gräupl, Corinna Schmitt, and Sophia Grundner-Culemann. "Security in Digital Aeronautical Communications A Comprehensive Gap Analysis". In: *International Journal of Critical Infrastructure Protection* 38 (2022), p. 100549. ISSN: 1874-5482. DOI: 10.1016/j.ijcip.

2022.100549. URL: https://www.sciencedirect.com/science/article/pii/S187454822200035X

10. Antonius Scherer, Tobias Guggemos, Sophia Grundner-Culemann, Nikolas Pomplun, Sven Prüfer, and Andreas Spörl. "OnCall Operator Scheduling for Satellites with Grover's Algorithm". In: *Computational Science - ICCS 2021 : 21st International Conference, Krakow, Poland, June 16-18, 2021, Proceedings, Part VI*. ed. by Maciej Paszynski. Vol. 12747. Lecture Notes in Computer Science. Cham: Springer, 2021, pp. 17–29. ISBN: 978-3-030-77979-5. DOI: 10.1007/978-3-030-77980-1_2

11. Korbinian Staudacher, Tobias Guggemos, Sophia Grundner-Culemann, and Wolfgang Gehrke. "Reducing 2-QuBit Gate Count for ZX-Calculus based Quantum Circuit Optimization". In: *Electronic Proceedings in Theoretical Computer Science* 394 (Nov. 2023), pp. 29–45. ISSN: 2075-2180. DOI: 10.4204/eptcs.394.3. URL: http://dx.doi.org/10.4204/EPTCS.394.3

2. Background

This chapter provides background knowledge about cryptography and formal methods. Specifically,

- Section 2.1 describes the concept of information security.
- Section 2.2 introduces (asymmetric) cryptography at beginner level.
- Section 2.3 compares the key management approaches PKI and IBC.
- Section 2.4 gives a definition of revocation and key change from the literature (which is challenged in Chapter 4).
- Section 2.5 delineates the difference between in-group communication and group communication.
- Section 2.6 gives an overview of standard attacker models used in security analyses.
- Section 2.7 introduces automated reasoning in general and trace-based reasoning in particular.

2.1. Information security

Whether or not a message is confidential or has an authentic source is a question of *information security*, a research field that studies certain properties of information and of its origin.

In the standard document "ISO/IEC 27000"[ISO18], the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) define seven such properties. Many information security goals can be achieved or supported with cryptography. As an example, consider the three properties that are most closely connected to our work:

Confidentiality *Property that information is not made available or disclosed to unauthorized individuals, entities, or processes*

Integrity *Property of accuracy and completeness*

Authenticity *Property that an entity is what it claims to be*

One can facilitate ...

... confidentiality by *encrypting* a message that should be confidential with a cryptographic key, so that only the intended recipient(s) can reverse the encryption (*decrypt*) and read the message.

... integrity by applying a mathematical function on the message that generates a forgery-resistant fingerprint of it. If the fingerprint is attached to the message in a secure manner, the recipient can re-calculate the fingerprint and compare, thus convincing themself that the message was not altered in transit.

... authenticity by *signing* a message in an unforgeable way. Any recipient with a corresponding verification key can convince themself that the signature is real and that the message comes from the stated sender.

In this work, the focus is on encryption- and signature mechanisms that use asymmetric keys. The next section introduces this concept.

2.2. Asymmetric Cryptography

There are two main flavors of cryptographic keys: Symmetric keys and asymmetric keys.

Symmetric keys allow the user to perform inverse cryptographic operations (encrypt – decrypt, sign – verify) with the same key. That also means that all parties to a conversation need to have the same key and that they need to obtain it before they can participate. However, in many settings the communication partners are not pre-acquainted, have no private channel to communicate and thus can not agree ex ante on the key they would like to use. Also, a signature generated with a symmetric key can not be attributed to an individual user, but only to the group of users who shares this key.

In both cases, *asymmetric cryptography* (or: public key cryptography) helps: *Asymmetric keys* come in pairs of one secret key (which only the owner has) and one public key (which is public knowledge). The keys invert each other in the following way: Consider communication partners Alice and Bob. Bob can encrypt a message with Alice's public key, but he can not decrypt it. Only Alice can decrypt the message, because only she knows the secret key. For authentication, Alice signs a message with her secret key and Bob (or anyone) can verify the signature using Alice's public key. There is a catch, however: Bob needs to ensure that he uses the correct public key and is not hoodwinked by an attacker who impersonates Alice and publishes a false key in her name. The next section discusses such key management questions.

2.3. Key management: PKI and IBC

Public key cryptography only works if Bob knows which key belongs to Alice. This problem is most commonly solved with a PKI as defined through the X.509 certificate standard RFC 5280 [Coo+08]. IBC is an alternative that aims to reduce the management overhead inherent in PKI. Each approach is explained in more detail below, PKI in Section 2.3.1 and IBC in Section 2.3.2.

2.3.1. Public Key Infrastructure (PKI)

In a X.509-Public Key Infrastructure, Alice's public key is bound to her identity via a *certificate*. She obtains it by registering her public key with a so-called CA, a third party who issues digital certificates of public keys to users. There are three steps:

1. **Alice chooses an asymmetric key pair.** This is done in such a way that the secret key is not (easily) computable from the public key. Usually this means that Alice chooses a secret key and derives the public key from it.

2. **Alice authenticates herself to the CA**, presents her public key and proves that she holds the corresponding secret key. Note that she does not need to disclose her secret key to the CA for this.

3. **In return, the CA signs and issues a certificate** stating that the public key in question belongs to Alice.

Other users who trust the CA will accept this certificate as proof. It is important to note that the CA is assumed to be trustworthy. Users who do not know or trust the CA will accept the certificate if the CA itself is certified by a higher-level CA or by a chain of higher-level CAs that each certifies their subordinates (*Chain of Trust*) up to a root-CA which the users do trust.

While this system is functional and well-established, the certificate management is expensive and cumbersome for users, because they need to download certificates and validate them before using the corresponding public keys. Also, in most cases, it is not enough to validate a certificate once, but one must re-validate it regularly to make sure that it has not expired or been revoked (e.g., because the corresponding secret key was stolen).

There are many variants of this basic idea that aim at mitigating concerns about trust, performance or security; for a short overview, consult [Bas+14].

2.3.2. Identity-based Cryptography (IBC)

A more straightforward way to connect identity and public key, known as Identity-based Cryptography (IBC), was proposed by Adi Shamir in 1984 [Sha85]: The idea is to mathematically derive Alice's public key from her identity, or more precisely: from the digital representation of her identity (e.g., her e-mail address). This setting requires that Alice can not derive the corresponding secret key herself; otherwise, anybody could derive the secret key from her identity (= her public key). A so-called TTP needs to compute the secret key for her. The TTP has an asymmetric key pair (the *msk* and the *MPK*) which it uses to derive the users' secret keys[1]. There are, again, three steps:

1. **Alice authenticates herself to the TTP** and they establish a secure connection.

2. **The TTP derives a (or the) user secret key (*usk*) for Alice** using the *msk* in such a way that *usk* is hard to derive from Alice's identity or the *MPK*.

3. **The TTP sends Alice her secret key** over the secure line so that only she learns it.

Other users who want to communicate with Alice need to know Alice's identity and the *MPK*. From this they can derive Alice's public key and encrypt messages or verify signatures with it. In other words, there is no need for double-checking the connection between Alice's identity and her public key: They are mathematically connected through the master keys.

[1] It is usually pre-supposed that the users trust the master key pair; thus, mechanisms of master key management are out of scope for this work.

There are proposals for both Identity-based Encryption (IBE) and Identity-based Signatures (IBS). A classic IBS scheme consists of the following four algorithms [GG09]:

Setup/Key-generation $\mathcal{G}$ outputs the master key pair (msk, MPK), depending on a security parameter λ. Other parameters and functions (e.g., mathematical groups, hashing functions) that are necessary for the specific scheme are chosen as well.

Extract $\mathcal{E}$ outputs a signing key usk for a given identity ID.

Signing $\mathcal{S}$ signs a message m from the set of all possible messages $\mathcal{M}$ using usk. The output is a signature $\sigma = Sign_{usk}(m)$.

Verification $\mathcal{V}$ verifies or falsifies that a signature σ corresponds to a given message m, the stated sender's Identity (ID), and the TTP's MPK.

In practice, IBC is scarcely used [Tel23], because it requires strong trust: Another party besides the user legitimately knows the private key. This is known as *key escrow*. Therefore, schemes are only secure under the strong assumption that the third party is trustworthy.

There are ideas for escrow-free IBE and IBS (see for example [LQ04; Zha+12; YSM10]).

To address key revocation in IBC, it is helpful to understand the basic concept, its implementation in PKI and the closely related concept of key change. The next section explains them.

2.4. Revocation and Key Change

The US-American National Institute of Standards and Technology (NIST) defines *key revocation* as "[a] possible function in the lifecycle of a cryptographic key; a process whereby a notice is made available to affected entities that the key should be removed from operational use prior to the end of the established cryptoperiod[2] of that key." [Bar20]

Revoking the public key of a user may be necessary if, for example, she leaves her job for which she used the key, or her secret key was compromised (i.e. definitely or possibly disclosed to an unauthorized party), or if she is identified as an attacker.

When unacquainted parties (i.e. parties that have not pre-shared any keys, do not know each other, or have not established any other trust anchor between each other) want to communicate using public key cryptography, they always need a third, brokering party like the CA or the TTP, not only for the first connection, but for revocation, too[3].

2.4.1. Revocation in PKI: Certificate Revocation Lists (CRLs)

In the X.509 standard PKI, key revocation is commonly solved with Certificate Revocation Lists (CRLs) (RFC 5280 [Coo+08]), which record certificates that users should no longer trust and whose associated public keys they should no longer use although they have not officially expired.

[2] In the same document, *cryptoperiod* is defined as "the time span during which a specific key is authorized for use by legitimate entities or the keys for a given system will remain in effect".

[3] The third party entities that issue certificates/keys and those who revoke them are not necessarily the same, but for the purposes of this work, all third party key management tasks are attributed to the same abstract instance. For details about the distinction, see [Gug20], for example.

This approach is cumbersome because it requires an ad-hoc online validation of certificates when a connection is (re-)established and usually the download of the entire list [Bas+14].

A faster option for validity checks in PKI is the Online Certificate Status Protocol (OSCP), where users can inquire online about the current status (revoked or valid) of a certificate. Still, the latency and resource cost for the client checking a certificate may be significant in both cases.

2.4.2. Key Change

If a user's key was revoked but they are still authorized to hold a key, a *key change*[4] may occur; NIST [Bar20] defines this as "the replacement of a key with another key that performs the same function as the original key". As possible key change methods it mentions

a) *re-keying*, where the replacement key is generated independently of the original key, or

b) *key update functions*, where the replacement key is derived from the original key.

While key updates may be easier to execute, National Institute of Standards and Technology (NIST) specifies that they are inadmissible in US-federal applications because an attacker who breaks one of the keys in the update chain also gains information about replaced or derived keys.

2.5. In-group communication vs. group communication

The use case of this work is one-on-one communication between any two users who use asymmetric cryptography for encryption or signing, and who trust the same key management authority to validate their counterpart's key. For some discussions in this work, it is useful to collectively address all users who trust the same key management authority and hold corresponding keys. We therefore treat the non-revoked users as a "group" and revocation as an exclusion from this group. In that sense, this work considers "in-group" communication between any two entities with a valid key.

However, this must not be confused with a discussion of group communication, where a message has more than one sender or recipient. Messages from a TTP may have several recipients, but they happen on the key management level, not the communication level for which keys are modeled. They are assumed to be confidential, authentic and integrity-protected a priori, so the group-communication aspect is not part of our work.

The terminology for the communication scenario considered in this work includes

users, clients: any entities on the communication level who may send messages, apply for key material or certificates, and act as adversaries.

members (of the group): the users that currently hold a valid key w.r.t. the trusted key management party; "former members" are those who previously held a key, which has since been revoked and not re-keyed nor updated by the trusted party. Adversaries may be (former) group members.

[4] Some sources also refer to it as *key renewal*.

2.6. Attacker models

The security of a protocol or mechanism is captured by two dimensions: What abilities the attacker is assumed to have and what the security goals are.

There are four basic attacker models in which to analyze cryptographic protocols. They can be ordered by the increasing ability they respectively attribute to an attacker[5]:

1. **Logics of knowledge and belief** The attacker is not concretely modeled at all and does not have any concrete ability. The model only examines the information that honest participants (i.e. participants that follow the protocol rules and do not try to break the security) can deduce from a successful protocol run (e.g., the message length if the underlying scheme produces a ciphertext that is as long as the message itself).

2. **Dolev-Yao model** The attacker is modeled as a (dishonest) message carrier who can intercept, delay, modify, replay, delete and create messages, and whose abilities are restricted to these operations (or a subset thereof). This model is named after the authors who introduced it [DY83].

3. **Computational model** The attacker model is restricted to polynomial-time computations and may include Dolev-Yao style abilities like message interception. That an attacker can not break the protocol or cryptographic scheme is proven by reduction, i.e. arguing that a successful attack on the scheme would imply a successful attack on a problem that is believed to be computationally hard. It is also known as the complexity-theoretic model.

4. **Information-theoretic model** In this model, the attacker has infinite computational power and may be modeled with Dolev-Yao-style abilities. Roughly speaking, information-theoretic security is achieved when an attacker can not recognize that she has the secret key even when she does.

2.7. Formal verification

Cryptography aims at security. As Bruce Schneier famously stated: “Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can’t break.” [Sch98; Sch11], meaning that people all too easily convince themselves that they have created an unbreakable cipher when they have not.

Putting cryptographic mechanisms or protocols under careful, many-eyed scrutiny is therefore standard procedure to gain trust in their security. Security claims are most often supported in the form of manual, pen-and-paper reasoning. Still, history is rife with attack vectors that went unnoticed for years despite rigorous inspection by third parties [NS78; VP17; CD23]; RFC 6151 [TC11]. Two reasons can be identified [Bar+21]:

1) Security proofs are typically tedious and complex. Spotting logical errors is notoriously hard even for the most skilled cryptographers.

[5] While these models are common understanding in IT security canon, the precise hierarchy is not; it appears in a lecture video “Cathy Meadows, ”Cryptographic Protocol Analysis” (2/9/04)” uploaded to YouTube by the “securitylectures” channel, showing a lecture including slides held by Catherine Meadows at the Naval Postgraduate School.

2) Correct proofs may be based on oversimplified (and unconscious) assumptions, which fail to catch certain attacks; again, such modeling errors or their implications can be hard to spot even for experts.

A famous example for the second kind of oversight is the Needham-Schroeder public key protocol [NS78]. The protocol proposes a simple mechanism whereby Alice and Bob establish two shared secrets by encrypting and exchanging "recently generated"[6] nonces. The security argument assumes that the participants do not leak secrets, and under this assumption, the argument is correct. However, if this assumption is relaxed for Bob, he can easily fool a third party Charlotte into thinking that she is communicating with Alice, while Alice rightfully believes she is communicating with Bob. Thus, a seemingly innocuous assumption can obscure a strong attack. Note that Alice is also fooled in one aspect: Bob masquerades Charlotte's nonce as his own in his interaction with Alice. Alice remains unaware of this and inadvertently decrypts Charlotte's nonce for Bob.

In this particular example, the flaw was detected with a computer-aided analysis [Low96]. It is one of the first and most prominent cases to illustrate how computer-aided formal verification methods (also referred to as "automated reasoning") support human reasoning [CM12].

Automated reasoning can also be deployed to analyze protocol implementations rather than design. There are three main levels [Bar+21]:

1. **Design-level security** means that the underlying mathematics of the mechanism is sound (computational security) and the protocol logic holds (symbolic security). For example, if the protocol aims at authenticating the peers to each other, the design-level check ensures that the logical design of the authentication is indeed effective. The tools designed for this feat are also called "model checkers" (because they model the logic of a process and prove certain statements about the model).

2. **Functional correctness and efficiency** of a program mean that the implementation faithfully translates the design and works efficiently. If the implementation deviates from the specification, e.g. by hard-coding numbers the protocol requires to be chosen at random, it can not rely on the security guarantees given on the design-level.

3. **Implementation-level security** means that an attacker can not learn more than they should when the implementation is deployed, e.g., through side-channel attacks (where an adversary observes execution times or other, similar side-effects of running a cryptographic program to deduce secret information).

This work addresses revocation mechanisms and their protocol logic, which belong to the design level. Symbolic model checking is explained in more detail below.

Symbolic model checking

Symbolic model checking abstracts the implementation details of a protocol and reasons about its structure and components. There are two flavors:

[6] The protocol relies on Alice to decide whether the nonce that Bob mirrors back to her was recently generated by herself, and vice versa for Bob.

- trace-based reasoning, where the properties of possible protocol runs are analyzed directly
- equivalence-based reasoning, where two possible protocol runs are compared to each other and an adversary should not be able to distinguish them

The following explanation only covers trace-based reasoning, because it is the focus of this work. For details about equivalence-based reasoning, check [Bar+21].

A symbolic model consists of the following:

Atomic symbols like keys, messages, nonces, etc., which can not be split into other components. This means that the symbolic model can not reason about the contents of a message, for example, or find attacks that are based on sending messages that are similar to previous ones. In the same way, relations between keys can only be modeled with specific functions or equations and usually do not rely on conventional bit- or number mathematics.

Cryptographic functions are defined as symbols with a certain arity. For example, the unary function $pk(\cdot)$ may symbolically express that Alice's $pk_A := pk(sk_A)$ is the corresponding public key for her secret key sk_A. Depending on the way these function terms are used (either in a process description or in the equational theory), the model captures specific characteristics of the function and allows the prover to derive other terms from it.

Equational theories are a means to capture the cryptographic meaning of functions. For example, the connection between symbols sk_X and $pk(sk_X)$ is captured by the following equation that simultaneously gives meaning to the function symbols $sign(\cdot,\cdot)$ and $verify(\cdot,\cdot,\cdot)$:

$$verify(m, sign(m, sk_X), pk(sk_X)) = true$$

It specifies that the function verify evaluates to $true$ if the second argument is a signature of the message m with a key sk_X, the verification is done with the public key corresponding to sk_X and the first arguments of $sign$ and $verify$ are identical. If no other equation allows $verify(\cdot,\cdot,\cdot)$ to evaluate to $true$, the implication holds both ways. The specific primitives are treated as black boxes and only their abstract role in the protocol is modeled; that means, "sign" is a stand-in symbol for any signature method as long as no further mathematical characteristics (e.g., a lattice problem or a Diffie-Hellman-problem) are modeled for it.

Program state transitions capture the possible steps in the protocol. States have timestamps, which allows ordering them, and each contains different information. For example, a state may contain a fact $hasKey(Alice, sk_A, pk(sk_A))$, which codifies that Alice owns a secret key sk_A and a corresponding public key.

Some information may be global, so that it appears in every state after a certain timestamp. When Alice has created her key pair, $hasKey(Alice, sk_A, pk(sk_A))$ should be a global fact to model that she can use her keys more than once.

Other information may be ephemeral, so that it is generated during one transition and can be consumed in a later one. For example, a fact *FirstMessageSent* may emerge as the result of an agent action *SendFirstMessage* and serve as the precondition for the

next step, *SendSecondMessage*. In this case, the agent action *SendSecondMessage* consumes the fact *FirstMessageSent* and the action *SendSecondMessage* can not happen again until *SendFirstMessage* is triggered again. This way, the order in which steps occur in the protocol is captured in the program.

These components allow the creation of *traces*, i.e. finite sequences that consist of any specified protocol steps. Traces can be checked for properties; for example, a sanity check for the model can ensure that at least one trace exists where the last step of the protocol is reached. If the order of the steps is modeled correctly, a successful sanity check means that a normal run of the protocol is possible in this representation.

Security properties are expressed as (conditioned) trace properties, e.g. saying that in all traces, if Bob seems to interact with Alice, he actually does, unless the attacker knows a certain key. A model checker reconstructs all possible ways to reach the specified state and may thus find traces that violate this claim. If all traces (or at least one trace, depending on what is claimed) fulfill the statement, the property is proven to hold.

Attacker model Symbolic reasoning tools most often model a Dolev-Yao attacker [Mea11]. It can execute all the same actions as any agent in the system, knows public values and might be able to leak certain information (depending on the exact model). The equational theory allows an attacker to derive new knowledge from any information it has access to. If a model checker finds an attack against the protocol (i.e. a trace that violates a security claim), the attack is evident from the trace.

3. Related Work

This chapter provides an overview and a discussion of related work. More precisely,

- Section 3.1 briefly outlines several approaches to IBC-revocation and illustrates their interrelation (Figure 3.1, explained below). A more detailed analysis and comparison follows in Chapter 4.
- Section 3.2 presents the current state of research about the formal verification of IBC and of revocation-like mechanisms.

There is no previous work on formal verification of identity-based revocation mechanisms.

3.1. Revocation in Identity-based Cryptography

The most influential contributions to the problem of identity-based revocation are found in four publications [BF01; BGK08; LV09; SE13b]. The following three sections outline these approaches in chronological order:

- Section 3.1.1 introduces "naïve" IBC-revocation as described in [BF01].
- Section 3.1.2 introduces non-interactive revocation as described in [BGK08] and [LV09].
- Section 3.1.3 introduces decryption key exposure attacks and -resistance as described in [SE13b].

The order serves to highlight the novelty each provides over the previous work; [BGK08] and [LV09] are similar and therefore introduced together.

Section 3.1.4 describes further work in this research area.

Figure 3.1 provides a systematic overview of certain related work as explained below. The [nodes] correspond to literature references. The central horizontal axis is ordered chronologically from left to right, starting with [BF01]. The graph highlights the following:

- The publications denoted in **black** text are included in the more detailed analysis in Chapter 4.
- The ones denoted in gray text appear in the illustration because they are the main references for other work, but are left out of the detailed discussion for the following reasons:
  - [Han+05], because their approach is the only one to use extra hardware for revocation, which makes it harder to compare to the others.
  - [TT12], because the construction is not as secure as originally claimed. The arrow label is therefore crossed out in the illustration.
  - [TTW13], because the construction relies heavily on [TT12].

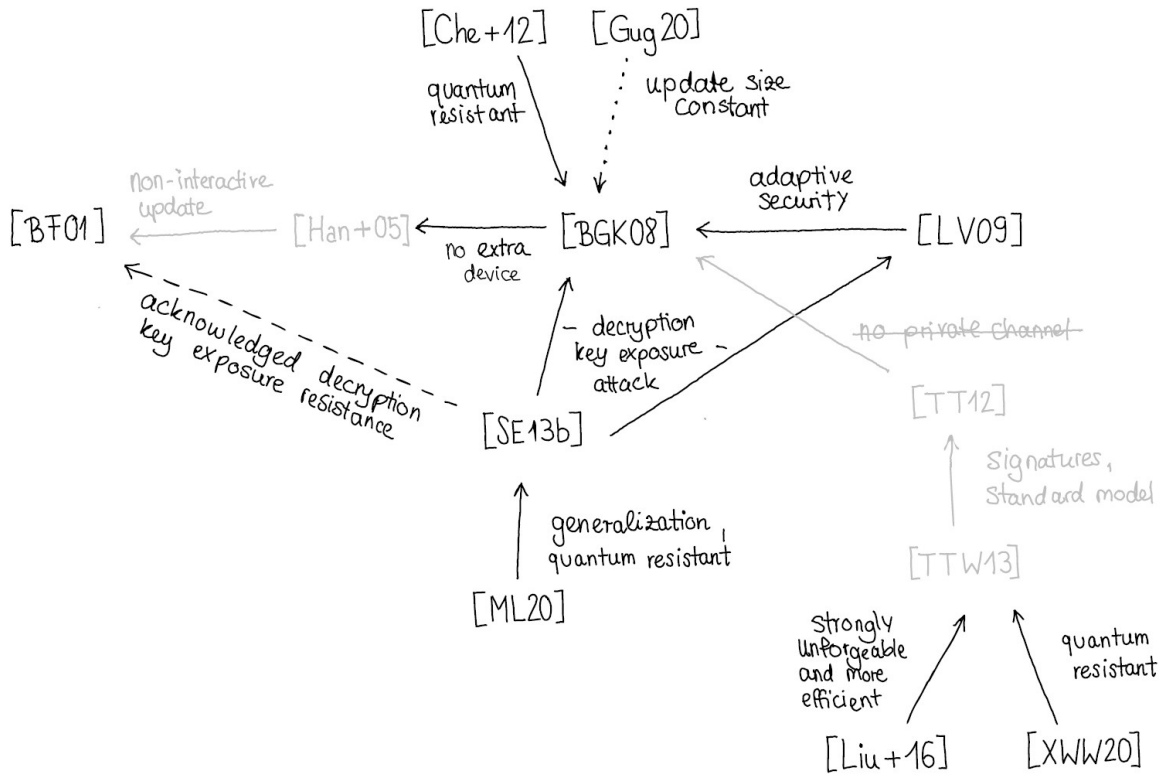

Figure 3.1.: Interrelation of related work on revocation in IBC: The nodes of this graph represent publications in which revocation in IBC is discussed, indicating the authors' names, year of publication and/or reference handle in the bibliography. An arrow from one publication pointing to another indicates that the first references the second as a main inspiration. Arrow labels indicate the most important difference that the first claims to make with respect to the second. (Note that the graph is not complete: Even if no arrow appears between publications, one might cite the other.)

- Solid arrows point from a chronologically later publication to another one it cites as main inspiration. The arrow labels state the pivotal difference between the two.
- The dashed arrow highlights the similarity to [BF01] explicitly acknowledged in [SE13b].
- The dotted arrow indicates the main difference between [BGK08] and [Gug20] to place the latter within the related work landscape even though it cites none of the other publications in this graph.

3.1.1. Boneh & Franklin, 2001: "Naïve" IBC-revocation

In their publication "Identity-Based Encryption from the Weil Pairing" [BF01], authors Boneh and Franklin propose the "first fully functional" IBE-scheme (a claim echoed in later publications [BGK08; ML20]) and include a first, very brief discussion of IBC-revocation. The authors observe: Since the public key is an arbitrary string, it can include a time stamp,

delineating different validity periods (which we also call "epochs"). Alice may, for example, encrypt a message under the public key "bob@example.edu ‖ *current-week*", which Bob can only decrypt with a private key issued for this e-mail-adress and the current week. Alice therefore does not need to keep track of certificate revocation lists or check whether Bob's certificate has expired. A subsequent publication [SE13b] refers to this approach as the "naïve" one because it is straightforward and not concerned with efficiency.

3.1.2. Boldyreva, Goyal & Kumar, 2008: Non-interactive Revocation

For N non-revoked users in a communication system, the naïve solution to revocation not only requires the TTP to compute N new keys but to send them to the users over N individual secret channels. Inspired by an approach where additional devices for each user allow the TTP to distribute updates over public channels (thus called "non-interactive") [Han+05], authors Boldyreva, Goyal and Kumar [BGK08] aim to achieve non-interactive revocation without extra hardware. They use two concepts:

1. Separation of every user's secret key into an identity-bound component and a time-bound component, in which the time-component can be updated.

2. A binary tree data structure to reduce, from linear to logarithmic, the number of key updates that the TTP needs to compute.

Because of the complexity reduction, the authors call their construction "(efficiently) Revocable Identity-based Encryption (RIBE)". This denominator is used for many subsequent approaches (e.g., [LV09; SE14; ML20; TTW12; TT12; Che+12; XWW20]). Mathematically, the construction relies on polynomials as secret keys. This ensures that update values that are published for one user are useless for other users, since the correct key can only be computed from an update on the matching polynomial. By choosing a suitable degree for the polynomial, a number of attributes (like "identity" and "time") can be controlled for in the key. By constructing a binary tree with polynomials assigned to each node and user secret keys (also polynomials) assigned to each leaf, the number of necessary updates becomes logarithmic instead of linear in the number of users: If a subset of users shares one of the higher-level polynomials because they are part of the same sub-tree and no user in this subset is revoked, the TTP can simply issue an update for this polynomial. Otherwise, updates for smaller subsets are necessary.

Libert and Vergnaud [LV09] present an important variant: Since [BGK08] only prove *selective-ID security* (i.e. only in a model where the attacker targets a specific ID), [LV09] shows that *adaptive-ID secure* RIBE (where the attacker can target IDs more flexibly) is also possible. Their proposal is based on a previous, revocation-free scheme. Since adaptive-ID security is deemed stronger than selective-ID security, the contribution of such a scheme is noteworthy.

3.1.3. Seo & Emura, 2013: Decryption key exposure attack and resistance

Seo and Emura [SE13b] contribute the following insight: If the secret key for a certain validity period ("decryption key" or "signing key") is computed from a static secret key and a key update in a reversible manner, and the key updates are broadcast, then exposure of the derived secret key may allow the computation of the static secret key. The authors

point out that [BGK08] and [LV09] are vulnerable to this attack. This was not noticed in the original security analyses because the attack happens outside their security models. The approach in [BF01] withstands the attack, since there is no static part of any secret key; keys are always replaced entirely. They propose a RIBE-scheme which combines the techniques underlying [LV09] with collusion-resistant Hierarchical Identity-based Encryption (HIBE)s and a re-randomization technique, which allows them to achieve *Decryption Key Exposure Resistance*. The scheme is adaptive-ID secure and at least as efficient as [BGK08] and [LV09].

3.1.4. Further work

The following sections introduce work that,

1. adds quantum-resistance.
2. discusses revocation in identity-based signature schemes
3. proposes an update with constant size
4. proposes master key renewal for revocation

Quantum-resistant schemes

Lattice-based schemes are assumed to withstand attacks from quantum computers, which may eventually break classical cryptography. A lattice-based RIBE-scheme was proposed in 2012 [Che+12]. It is based in part on the binary-tree data structure from [BGK08] and derived from another lattice-based IBE-scheme [ABB10]. More proposals for (revocable) lattice-based IBE exist (e.g. [Shw+11; CZ15; TW17; TW21]). The open problem of deriving a RIBE- from any IBE-scheme [SE13b] is solved with a generic construction that allows for an adaptive-ID secure, lattice-based and decryption-key exposure resistant IBE scheme [ML20] *(bottom row, left, in Figure 3.1)*.

Signature schemes

There also exist revocable identity-based schemes for signatures (RIBS): The preferred security notion for signature schemes to fulfill is *strong unforgeability*, which is achieved by a construction based on bilinear pairings and a binary tree data structure [Liu+16] *(bottom row, center, in Figure 3.1)*. The work improves a previously proposed signature scheme [TTW13] that is strongly based on another encryption scheme [TT12]. Because the publication for the latter erroneously claims to improve [BGK08], as pointed out in [SE13a], [TT12; TTW13] are not considered here in detail. There is also a quantum-resistant RIBS-scheme [XWW20] based on lattices that achieves the weaker *existential unforgeability* notion *(bottom row, right, in Figure 3.1)*.

Constant update size

Signatures are also the focus of an approach that improves the update efficiency compared to [BGK08] from logarithmic to constant [Gug20] *(top row, left, in Figure 3.1)*: Instead of a split secret key where the time-component may be updated periodically, the approach changes the msk that underlies all keys. If msk is an element of a cyclic group, multiplying it with

a random group element Δ yields a new value that seems random without the knowledge of Δ. Under certain circumstances, this change can be translated to the users' key by updating each of them with Δ. Using a binary-tree data structure for the private channels between TTP and users allows the update to be distributed comparatively efficiently even though the update can not be broadcast (or revoked users would be able to update their keys, too).

Master key renewal

Another work([gGG18], see Section 1.5) proposes the complete renewal of the master key material (i.e., a fresh instantiation) to achieve revocation. This implies the complete renewal of all user key material. While this approach is effective, the dependencies between different instantiations of an IBC-scheme are outside the scope for this work; this work only considers systems in which keys of different epochs are at least somewhat mathematically related.

3.2. Formal verification of IBC and revocation approaches

The following two sections describe work on

- formal verification of IBC protocols and related revocation problems
- formal verification of revocation problems outside of IBC

3.2.1. IBC and IBC-revocation

There is little research about formal verification of identity-based protocols. Three publications stand out:

- The comprehensive study "Symbolic Analysis of Identity-based Protocols" [BHS19] is the earliest and most important contribution. It discusses how identity-based encryption and -signature schemes can be abstracted in such a way that the trace-based automated prover Tamarin can reason about them. The paper provides both, a simple, highly abstracted model for IBC and a mathematically more precise model that includes details of so-called bilinear pairings. Bilinear pairings are an important ingredient of many identity-based schemes, especially early ones (see for example [BF01; Hes02; Hes03; LQ03; Bae+04]). The authors of [BHS19] find two things:
  1. Non-trivial logical attacks are found even by their highly abstract model; hence, employing a verification tool such as Tamarin is, in fact, useful for understanding identity-based mechanisms.
  2. Models that try to capture the algebraic intricacies of identity-based schemes quickly outgrow the capabilities of Tamarin because reasoning with distributive laws is still an open problem for symbolic tools in general. Since identity-based cryptography relies on distributive laws even when it avoids pairings (e.g., [GG09]), algebraically detailed analyses are difficult.

  Revocation is not addressed in this publication.
- A recent proposal for cloud-based payment systems [AB22] using identity-based signatures specifically discusses revocation and provides a security- and correctness analysis using ProVerif [Bla16], another symbolic tool. The proposed mechanism adds a time

key to the users' private key material. By updating the time keys, users can be revoked indirectly by excluding them from the update. The formalization models time keys; however, the corresponding security discussion remains superficial. Apart from an "ability to revoke a key", there is no specific analysis of the revocation mechanism, let alone dependencies between validity periods.

- Another claim to model and prove the security of an IBE-based Key Exchange Protocol using Tamarin is made in [SVR21]. However, the code uses a restriction ("OnlyOnce()") from the Tamarin manual in such a way that no session can established and the lemma is trivially true. Therefore, the analysis is futile.

3.2.2. Other revocation problems

Revocation has been formally analyzed in various publications. Most of them focus on proving that the mechanism for making the revocation known in the system is correct and sound. Only few proposals consider dependencies between keys of different epochs (mainly because in certificate-based approaches, there is none.)

For example, a formal analysis of vehicle-to-everything (V2X) revocation protocols using Tamarin [Whi+17] shows that their approach (adding a key pair for revocation to disguise the long-term identity of vehicles) allows revoking vehicles while hiding their long-term identity from other users. The scope ends at the revocation: There is no discussion (let alone verification) of how revoked vehicles might be re-keyed afterwards nor of any implications that a re-key might have for the protocol. The keys are managed through certificates, so no mathematical dependency between epochs needs to be considered.

Another certificate-based approach [Bas+14] goes further and specifically models not only revocation, but also key updates to verify the proposed "Attack-resilient PKI". But, again, as the infrastructure is based on certificates, there is no mathematical connection between validity periods to be considered, and the security properties discussed for it pertain to connection integrity, registration and legitimate updates - to certificate management issues, in short.

Closer to IBC-revocation approaches are the Double Ratchet (DR) formalizations. The DR protocol is at the heart of many secure messaging apps (e.g., the Signal App, WhatsApp, and the Facebook Secret Conversations [CCA23]). It employs two so-called ratchets, the public-key ratchet and the symmetric-key ratchet. The goal is to encrypt every message with a new key: The public key ratchet serves to derive the seeds for each epoch of the symmetric ratchets. From this seed, as long as the epoch continues, a new key is derived for every message, both by the sender (who encrypts the message) as well as the receiver (who decrypts the message).

The Signal App's session management, which includes the Double Ratchet, was formally analyzed using Tamarin [CCA23]; here the focus is on ensuring that the security guarantees that X3DH and DR provide for a session between two clients can be translated to a whole conversation (which is made up of many sessions). A mathematical connection between keys is considered because session keys are sequentially derived from each other through hashing. The model abstracts from detailed formalizations of the key exchange mechanism X3DH and the double ratchet to make the analysis tractable [CCA23].

The historic progression from Boneh and Franklin's "naïve" approach to more complex schemes using structures like binary trees highlights that IBC-revocation is not straightforward. Chapter 4 provides a detailed analysis of select schemes, distilling the important characteristics to compare and contributing high-level insights missing from the literature. Chapter 5 and Chapter 6 close the research gap between IBC-revocation and formal verification methods apparent from this chapter.

4. Assessment and classifications for IBC-revocation mechanisms

This chapter discusses the schemes highlighted in Figure 3.1 and various properties they have. It illustrates commonalities and differences between the schemes to highlight the research gap. It also gives rise to a systematization of revocation mechanisms in general and a classification of IBC-revocation approaches in particular. Specifically, the insights from this chapter are:

a) The security notions considered in the various publications are not universal and rarely include Forward- and Post-Compromise security, although these are established notions (Section 4.1).

b) Revocation in general can be achieved along two axes (Section 4.2)
   i. directly or indirectly
   ii. explicitly or implicitly

c) The known approaches to revocation in identity-based cryptography all work implicitly and indirectly, namely in one of three ways: By key renewal, with an individual update token, or with a universal update token (Section 4.3).

4.1. Characteristics and comparison

Proposals for identity-based revocation mechanisms typically discuss properties and characteristics that the respective authors deem important. While there are standard properties considered in most work, others vary between the papers. This leaves open questions.

The following list is a collection of characteristics from the publications in Figure 3.1:

- Encryption or signing algorithm
- Strategy
- Generic approach
- Static secret key part
- Re-randomization
- Update size
- Binary tree data structure
- Update distribution
- Decryption/Signing Key Exposure Resistance
- Forward Security
- Post-Compromise Security
- Collusion Resistance

Each is explained briefly below, together with a corresponding assessment of the publications.

Table 4.1 is a comprehensive overview of the assessments. It shows which characteristics the mechanisms are claimed to have (✓) or not have (✗).

The following qualifiers apply:

- Claims made by the authors in the specified publication itself are unmarked.
- Claims made by others are marked with a superscript symbol indicating the source (see legend).
- Empty cells indicate that neither the publication nor any related work addresses the property for the proposed scheme, and so does the entry "?".

A "?" shows where our work contributes an answer: The model proposed in this work helps understand the security properties of the different mechanisms.

The publications [BGK08], [ML20], and [Gug20] propose several mechanisms. The following are considered in the table:

- for [BGK08], the scheme presented their Section 4, "Main construction"
- for [ML20], the scheme presented in the section "A Generic Construction of RIBE with DKER"
- for [Gug20], the generic approach described in their Section 4.4.3, "Key Updatable Signature Scheme (KUSS)"

The following paragraphs explain the characteristics and the assessments. The corresponding row labels in Table 4.1 are denoted by **[label]**.

Encryption or signing algorithm [enc/sig] Every mechanism discussed in this chapter is either focused on encryption (entry "E" in the table) or signing (entry "S" in the table).

- Encryption: [BF01; BGK08; LV09; Che+12; SE13b; ML20]
- Signing: [Liu+16; XWW20; Gug20]

Strategy [re-key/token] Two strategies appear in the considered publications: Re-keying with an entirely new key, and sending update tokens for the user to compute new keys from their old ones.

- Re-key: [BF01]
- Token: [BGK08; LV09; Che+12; SE13b; ML20; Liu+16; XWW20; Gug20]

Generic approach [gen] Some approaches are generic: Even though they may be exemplified in a specific mechanism, they are applicable to (a subset of) other IBC-mechanisms that have no inherent revocation mechanism.

- claimed generic: [BF01; SE13b; ML20; Gug20] Of these, [BF01] is noteworthy because the proposed mechanism is so generic that the publication does not even discuss it at length: Choosing time-dependent identity values works for any IBC-scheme and will have the same security as the scheme provides for separate identities. The other three publications exemplify their approach but also describe it as transferable to further schemes.

Table 4.1.: Characteristics and properties of identity-based cryptography schemes: Each entry indicates whether the publication *(column)* claims that their mechanism does (✓) or does not (✗) have a property *(row)*. Empty cells and ? both indicate that the property is not definitively addressed in the publication itself or any related work item; where a ? appears, the model presented in this work provides answers (see Table 7.4).
References in the table: $*$ = [SE13b] † = [ML20]

Properties	[BF01]	[BGK08]	[LV09]	[Che+12]	[SE13b]	[ML20]	[Liu+16]	[XWW20]	[Gug20]
enc/sig	E	E	E	E	E	E	S	S	S
strategy	re-key	token	token	token	token	token	token	token	token
gen	✓	✗			✓	✓			✓
static *sk*	✗	✓	✓	✓	✓	✓	✓	✓	✗
re-rand.		✗*	✗*	✗*	✓	✓	✓	✓	
upd. size	$\mathcal{O}(n-r)^\dagger$	$\mathcal{O}(r \log \frac{n}{r})^\dagger$	$\mathcal{O}(r \log \frac{n}{r})^\dagger$	$\mathcal{O}(r \log \frac{n}{r})^\dagger$	$\mathcal{O}(r \log \frac{n}{r})^\dagger$	$\mathcal{O}(r \log \frac{n}{r})$	$\mathcal{O}(r \log \frac{n}{r})$	$\mathcal{O}(\log r)$	$\mathcal{O}(1)$
bin. tree	✗	✓	✓	✓	✓	✓	✓	✓	✓
upd. dist.	$\mathcal{O}(n-r)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n-r))$
D/SKER	✓*	✗*	✗*	✗†	✓	✓	✓	✓	?
Forw. Sec.	?	?	?	?	?	?	?	?	✓
Post-C. Sec.	?	?	?	?	?	?	?	?	✓
Coll. Res.	✓	✓	?	?	✓	?	?	?	?

- not claimed to be generic: [LV09; Che+12; Liu+16; XWW20]
- claimed to be not generic: [BGK08]

Static secret key part [static *sk*] Some approaches solve the revocation problem by constructing a two-component secret key, where one component - the secret key for the identity - remains fixed, and one component - the secret key for the date - changes. In these cases, there exists a static part of the secret key (namely the part associated with the identity), which is the same in all epochs, and updates are generated only for the time component.

This distinction leads to the use of separate terms in the literature: The static part is called *secret key (sk)*, the "full" key with which to decrypt (sign) messages is called *decryption key (dk)* (*signing key (sigk)*). To derive the decryption (signing) key of a given epoch, both components are needed. This work uses the terms "static user secret key" (*susk*) instead of *sk* and "user secret key" (with which to decrypt or sign; *usk*) instead of *dk*/*sigk*.

- static secret key part used: [BGK08; LV09; Che+12; SE13b; ML20; Liu+16; XWW20].
- no static secret key part used: [BF01; Gug20] In [BF01], the user's identity (= public key) includes a date and the secret key is bound to the time component. However, secret keys for the same user at different dates are not related.

Re-randomization [re-rand.] If the decryption key is leaked and the decryption key derivation is invertible, then the attacker can sometimes derive the *susk* from the decryption key. To avoid this, one can "re-randomize" the decryption key when deriving it, such that an attacker can not learn *susk* from the leak.

- re-randomization used: [SE13b; Liu+16; ML20; XWW20] While [ML20] and [XWW20] do not call their technique "re-randomization", only a derived, time-dependent form of the static keys contributes to the decryption key in their approaches. We interpret this as re-randomization.
- no re-randomization used, as pointed out by [SE13b]: [BGK08; LV09; Che+12]
- re-randomization not applicable: [BF01; Gug20]

Update size [upd. size] An important goal for "efficient" revocation approaches is reducing the size of the update which the TTP needs to compute in relation to the number of (non-revoked) users $n - r$ (where n is the number of most recent members, of which r are revoked and should not receive an update).

- linear amount ($\mathcal{O}(n - r)$) of updates: [BF01] Every non-revoked user requires an individual new key.
- logarithmic amount ($\mathcal{O}(r \log \frac{n}{r})$) of updates: [BGK08; LV09; Che+12; SE13b; ML20; Liu+16] All of them use the same binary tree-data structure approach to achieve this reduction (see also "Binary tree data structure").
- logarithmic amount ($\mathcal{O}(\log r)$) of updates: [XWW20]
- constant amount ($\mathcal{O}(1)$) of updates: [Gug20] The update token chosen at random by the TTP is the same for all participants.

Binary tree data structure [bin. tree] Many approaches under consideration use a binary tree data structure for efficiency gains: The nodes of the tree are associated with information that goes into the derivation of the decryption (signing) key. Leaf nodes are associated with individual users and a user's key is derived from all components assigned to the nodes on the path from the tree root to their corresponding leaf. If a user is revoked and should not receive an update for the next epoch, the TTP computes the smallest set of nodes $KUNode$ that does not contain a node on the path to any revoked user. Then it computes and distributes updates for all nodes in this set ("$KUNode$ algorithm"). [Gug20] uses the LKH (RFC 2627 [WHA99]) to gain efficiency: Here, the nodes are associated with (separate) keys to establish secure (broadcast) channels. When a user is revoked (= should not receive an update for the next epoch), the TTP computes the smallest set of nodes that does not contain a node on the path to any revoked user, and encrypts the update token (which is the same for all non-revoked users) with each key in the set separately. Thus, each non-revoked user can decrypt one of the resulting cipher texts and receive the update token.

- Binary tree data structure used: [BGK08; LV09; Che+12; SE13b; Liu+16; XWW20; ML20; Gug20]
- Binary tree data structure not used: [BF01]

Update distribution [upd. dist.] When an individual update is sent to every user over an individual secure channel they have with the TTP, the update distribution complexity is linear in the number of non-revoked users $n - r$.

When updates are computed with the $KUNode$ algorithm, the set of updates is useless for any user whose path in the tree does not contain a corresponding node. Therefore, the update set can be broadcast or publicly posted [BGK08; LV09] and no secure channels are required. This is called a "non-interactive" update [Han+05; BGK08]. The update distribution complexity is then constant ($\mathcal{O}(1)$), assuming that publishing one update has the same cost as posting several updates.

As explained above, a secure broadcast is possible via Logical Key Hierarchy (LKH) when the update token is the same for all non-revoked users. The complexity of this update distribution is logarithmic in the number of non-revoked users ($\mathcal{O}(\log(n-r))$).

- linear distribution complexity ($\mathcal{O}(n-r)$): [BF01]
- constant distribution complexity ($\mathcal{O}(1)$): [BGK08; LV09; Che+12; SE13b; Liu+16; XWW20; ML20]
- logarithmic distribution complexity ($\mathcal{O}(\log(n-r))$): [Gug20]

Decryption/Signing Key Exposure Resistance [D/SKER] When Decryption Key Exposure Resistance (DKER) is achieved, the leak of a decryption key does not allow the adversary to infer other secrets from it [SE13b]. To capture this security notion, it is necessary to distinguish the leakage of static key parts and leakage of the full decryption key.

[Liu+16] and [XWW20] naturally infer the property of Signing Key Exposure Resistance (SKER).

- claimed to achieve DKER/SKER: [BF01; SE13b; ML20; Liu+16; XWW20] [BF01] trivially achieves it, the others achieve it by re-randomization.

- claimed to not achieve DKER/SKER: [BGK08; LV09; Che+12] The failure to achieve this property went unnoticed because the security models are incomplete.
- no claim: [Gug20]

Forward Security [Forw. Sec.] For encryption, Forward Security means that messages encrypted with an old key stay confidential if a new key (e.g. an updated version of the old one) is leaked [BG21]. For signatures, it means that a leaked key from one epoch does not allow an adversary to sign any message from an earlier epoch [BG21].

- claimed for: [Gug20]
- no claims: [BF01; BGK08; LV09; Che+12; SE13b; ML20; Liu+16; XWW20]

[Han+05] points out that the secure channels for distributing the new keys in [BF01] should be forward secure, but does not apply this concept to the revoked keys themselves. Since only one of the considered publications discusses this property, it is not deemed a standard security notion for IBC-revocation.

Post-Compromise security [Post-C. Sec.] For encryption, Post-Compromise Security means that messages encrypted with a new key stay confidential if an old key is leaked. For signatures it means that the leak of an old key can be "healed" so the adversary can not sign messages in future epochs [CHK21].

- claimed for: [Gug20]
- no claims: [BF01; BGK08; LV09; Che+12; SE13b; ML20; Liu+16; XWW20]

Since only one of the considered publications discusses this property, it is not deemed a standard security notion for IBC-revocation.

Collusion Resistance [Coll. Res.] In [BGK08], the authors observe that " [...] in the IBE setting a revoked user, or the adversary holding its [secret] key, should not be able to decrypt messages even if it colludes with any number of non-revoked users."

- claimed for: [BF01; BGK08; SE13b] In [BF01], the introduction states indirectly that a "fully satisfactory" solution must not prohibit collusion. We interpret the absence of any further discussion as a claim that the proposed scheme allows it. For [BGK08], the authors claim that collusion is not useful because the updates are tailored to the individual or shared polynomials used as keys. Thus, revoked users can not use any of the available update tokens, because none of them fits any of the user's key parts. [SE13b] claim their mechanism inherits Collusion Resistance from the underlying scheme.
- no claims: [LV09; Che+12; ML20; Liu+16; XWW20; Gug20]

Summary

The following observations from the table are important for this work:

- The commonality of all considered IBC-revocation schemes is a reliance on re-keying or key updates, called "key change" methods by NIST. This deviates from the NIST definition of "revocation" (see Section 2.4) but is consistently called "revocation" in IBC-literature. Section 4.2 addresses this incongruity further.

- All publications except two ([BF01; Gug20]) use static keys in their approaches. Incidentally, the two exceptions also significantly deviate from the other publications with respect to the update size and distribution complexity: Update size is significantly more complex for [BF01] and significantly less complex for [Gug20] than for the others, and the distribution complexity is significantly more complex for both than for the others. These are systematic differences, which partly answers **RQ 1)**.

- Most publications consider Decryption Key Exposure Resistance, but do not discuss Forward- and Post-Compromise Security. This raises the question whether the three notions can be reconciled and if discussing either DKER or the other two is sufficient.

- Only two publications ([BGK08; SE13b]) discuss Collusion Resistance, though the concept is applicable to all other mechanisms as well.

The next section addresses the systematic commonalities of IBC-revocation schemes by classifying it within a broader systematization of key revocation methods (Section 4.2). Section 4.3 addresses the systematic differences and explains the classification of IBC-revocation schemes in three classes.

4.2. Two dimensions of key revocation

Recall the NIST's definition of key revocation from Section 2.4: "A possible function in the lifecycle of a cryptographic key; a process whereby a notice is made available to affected entities that the key should be removed from operational use prior to the end of the established cryptoperiod."

Compared to related work on IBC-revocation, this definition seems quite narrow: None of the described schemes requires that any notice be made available at all about the validity of any key or that the so-called cryptoperiod must end prematurely.

In our broader, approach-agnostic understanding, revocation is any process by which an authorized key management party renders a cryptographic key permanently useless and invalid.

Which strategy is chosen to revoke keys, depends on specific use cases and their requirements. We recognize two core dimensions to consider:

Dimension 1: indirect vs. direct This dimension describes whether a revoked key is handled directly and with actions that are directed at this key (like an announcement about a specific key) or indirectly, by taking some key management measures that are directed at non-revoked keys (like re-keying legitimate users).

Dimension 2: implicit vs. explicit This dimension describes whether users who want to use someones else's public key need to check its validity explicitly or not.

The revocation mechanism is *explicit* if users who encrypt a message or verify a signature need to separately ensure in an extra step that the public key in question is valid.

Revocation is handled *implicitly* if users can rely on some inherent mechanism to control the key validity and do not have to check it. In this case, the system is set up in such a way that the owner of a revoked public key can no longer use the corresponding

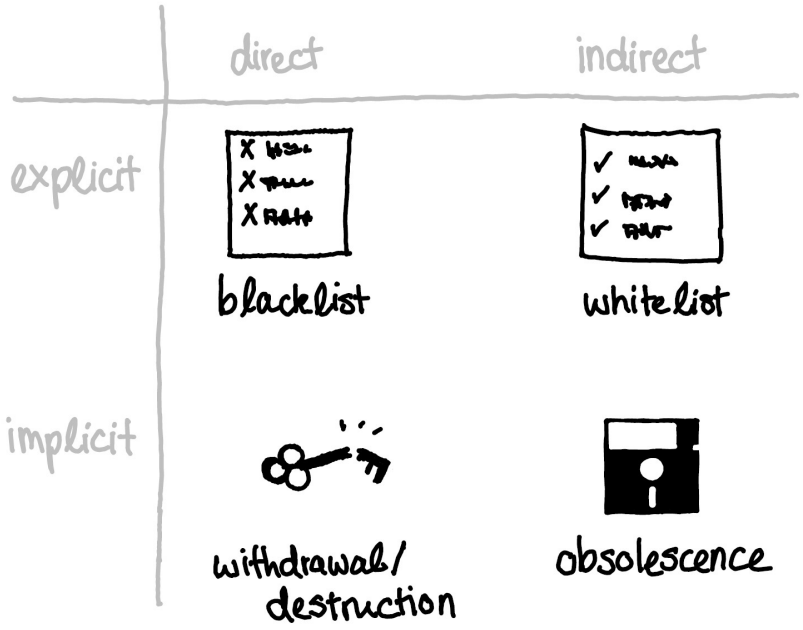

Figure 4.1.: Invalidating a key can happen through measures that affect the key directly or indirectly, and which are explicit (and externally overrule the key's properties) or implicit (where one of the key's inherent properties determines its use).
A direct, explicit measure is blacklisting, which prohibits the use of otherwise valid keys. As the indirect counterpart to blacklists, whitelists explicitly allow the use of certain keys, and non-listed keys may not be used.
Direct, implicit key validation is achieved by taking away the key from its owner.
Indirectly, a key can implicitly be invalidated by changing the cryptographic system in such a way that the key becomes obsolete.

private key with new messages. Thus, as long as all users adhere to the most recent specifications of the system, it is impossible to use a public key illegitimately in this setting, because it is not the sender's or verifier's responsibility to check the validity. Note that in this case, users may inadvertently send encrypted messages to revoked parties who can not decrypt it: They have no way to know whose keys are valid or not.

The two dimensions can be considered orthogonal, yielding four possible combinations (see Figure 4.1).

From top left to bottom right of Figure 4.1 they are:

explicit and direct This category can be described with *blacklisting*: The validity of a key or certificate is handled by the user checking a list that specifies all keys that have been deemed invalid. A key that is listed there should not be used. Hence, key revocation is handled by including the key in question in the updated version of the list (direct) and the list is an extra document referring to the key (explicit). This category is also the closest to NIST's requirement that a notice be made available to retract the key from use before the end of its cryptoperiod. Classic expiration mechanisms that define

cryptoperiods can also be direct and explicit, if they are included in a certificate; then they are specific to a certain key and need to be checked before use.

explicit and indirect This category can be described by *whitelisting*: The validity of a key (or the corresponding certificate) is handled by the user checking a list that specifies all currently valid keys (or certificates). No key that is not expressly listed there should be used. Hence, key revocation is handled by leaving out the key from the updated version of the list (indirect), which a user will check before employing the key (explicit).

implicit and direct This category can best be illustrated with physical keys. Consider that Alice has a padlock (= the public key) and a corresponding physical key (= the secret key). An implicit and direct way of revoking this key material would be taking away the key from Alice. This way, the padlock may still be used by anyone and it is not their responsibility to check whether Alice is allowed or able to unlock it (implicit) and her key can be removed (direct). Non-revoked keys are not affected by a revocation and remain usable. This category is unlikely to appear in the digital realm, because it requires the direct removal of a secret key that has previously been in use, including any copies. However, digital data that has once been released can afterwards hardly be removed from the entire system with certainty.

implicit and indirect This category can also be illustrated with physical keys: Consider that both Alice and Bob have a padlock (= public key) and corresponding key (= secret key) each. An implicit and indirect way of revoking this key material would be changing the admissible type of locks (e.g., from a padlock to a combination lock) as soon as Alice's keys are revoked. If all participants in the system are notified of this change, honest users will not use padlocks any longer, but switch to combinations locks, which renders Alice's and Bob's corresponding keys useless. Admissible locks can be used by anyone without concern (implicit), because it is not their responsibility to check whether Alice has a fitting key or knows the combination. Revoked keys are affected only indirectly because the circumstances of key use have changed. New, non-revoked keys need to be distributed and the "old" keys become obsolete (indirect). As [BF01] suggests, time stamps can be used as part of an implicit and indirect measure to synchronize the switch. A system switch could also be announced and synchronized by the TTP more spontaneously; as long as all users can reliably be informed about it, the synchronization does not need to happen through timestamps or in another (long) foreseeable manner.
Note that [BF01] interpret the time stamps as expiration dates. While that is a valid description, this work only interprets those revocation mechanisms to be "expiration-based" where the dates are treated as external information about individual keys, not inherent components of the encryption/signature scheme. In this sense, expiration-based mechanisms are categorized as direct, explicit revocation.

Using this systematization, it is easy to understand why none of the IBC-schemes in Table 4.1 uses revocation lists:

Recall that a central goal of identity-based cryptography is reduced key management effort, especially avoiding certificates and similar documents regarding keys. Explicit revocation techniques with which to communicate key validity statuses are therefore at odds with the basic idea of IBC. Since implicit, direct revocation techniques are unsuitable for the digital realm, the only solution for IBC-revocation are implicit, indirect approaches.

Similar statements appear in the literature: For example, Boldyreva et. al. point out that since "[...] there is no way to communicate to the senders that an identity has been revoked, such a mechanism to regularly update users' private keys seems to be the only viable solution to the revocation problem" [BGK08]. Others observe that "it is impossible, based on the conventional IBE model, for the user to immediately revoke and renew his decryption key only at times he needs to renew [it] without losing the advantage of IBE in terms of communication cost [...]" [Han+05].

The general systematization of revocation approaches given in this section fully supports these sentiments. It clearly shows that when the connection to an owner's identity is an inherent property of a key, the most consistent way of revoking them is by implicit, indirect means.

Note that revoking, in any way, the identifying information of a user from which the public key is derived (i.e. the e-mail-address in the example above) is not discussed in any related work. Without this option, the key-renewal and updates must inherently reference some information that distinguishes valid and revoked (or not-yet-valid) keys while the connection between identity and master key material remains. Most related work uses timestamps as an example for this information, and for good reason: Time stamps change continuously, in a foreseeable and unambiguous manner, and are available to all users.

Despite this commonality in the general approach to revocation, there are three main flavors of handling the time component in detail. The following section explains these differences.

4.3. Three revocation classes for identity-based cryptography

Even though all identity-based revocation mechanisms can be classified as implicit and indirect, there are still substantial differences. Table 4.1 shows them between three groups of publications, where the two mechanisms that do not use static keys also differ from the other ones in update size and -distribution complexity (and also differ between each other substantially). Indeed, these can be categorized in three main approaches:

- key renewal
- update with individual update tokens
- update with a universal update token

This section outlines the main characteristics of each approach and how they affect efficiency and security, with a summary of these insights in Table 4.2. The middle column of this table summarizes the main characteristics of the approach, and the right column indicates related work discussed in Section 3.1 that falls into the respective category.

4.3.1. Key renewal

This approach treats the time tag as a fixed part of the users' identities. The system specifies which time tag to use at which time, i.e. which tag-enhanced "identity" to use[1]. In this logic, revocation means that all users acquire "new" identities at the beginning of each new

[1]Note that the tag does not have to be time descriptor, as long as it is clear which tag to use at what time.

Table 4.2.: The three revocation categories for IBC have typical characteristics by which approaches can be classified. The right-most column cites approaches that fall into the respective class.

class	characteristics	corresponding literature
key renewal	• new keys independent of old keys • distribution complexity linear • computation complexity linear • MPK unchanged • attack vectors: *msk*, *usk*	[BF01]
individual update token	• dependence betw. old and new keys • distribution complexity constant • computation complexity logarithmic • MPK unchanged • attack vectors: *msk*, *usk*, *susk*, key update	[BGK08; LV09] [Che+12; SE13b] [Liu+16; XWW20] [ML20]
universal update token	• dependence betw. old and new keys • distribution complexity logarithmic • computation complexity constant • MPK changes • attack vectors: *msk*, *usk*, key update	[Gug20]

epoch. All "old" identities become obsolete, and all non-revoked users receive keys for their "new" identities via a private channel.

Approaches belong into this class if, in every epoch, the users are supplied with individual new key material that they can use as is, without putting it in the context of their old key material.

The class is characterized as follows:

- The old and the new key are completely independent of each other and of other users' keys.
- Because of this independence, there seems to be no option to distribute the keys in a more efficient way than over individual private channels[2] and there are no apparent benefits from using any tree effects to reduce the linear distribution complexity for the TTP.
- In the same way, the TTP must freshly compute all remaining users' keys after a revocation. The computation complexity is linear.
- MPK does not change across epochs.
- The attack vectors are *msk* and *usk*. Those are values to keep secret from the adversary.

[BF01] is an example for this class.

[2]It can be argued that key distribution does not need a private channel, because the encrypted values can be published; however, each key still needs to be encrypted separately, so the 1:1-complexity of the problem remains.

4.3.2. Individual update token

This approach treats the time tag as one of several aspects of an identity (similar to attribute-based encryption techniques [SW05]) in such a way that the key can be updated for every epoch. Instead of computing and distributing entirely new key material, an update token is generated for each user individually with which they can update their secret key locally.

Approaches belong into this class if, in every epoch, the users are supplied with an update token for their key material that they have to apply to their old key material and if this update is different for at least two non-revoked users in the system (although some users may get identical updates for a shared portion of their key material), such that a revoked user could not apply any of the distributed update tokens to their old key to get an update.

The class is characterized as follows:

- The local update mechanism creates a dependency between the old and the new key.
- Because the individual updates are, in themselves, useless to an attacker who does not hold the corresponding secret key, they can be published. This allows the distribution complexity for the TTP to be constant.
- The computation complexity is logarithmic if a certain key hierarchy is used (which all related schemes do). In that case, each user holds several keys. Details are omitted here; refer to [BGK08] for more information.
- *MPK* does typically not change across epochs.
- Besides *msk* and *usk*, *susk* and the update tokens are additional attack vectors, as explained in Section 4.1.

Examples for this class are [BGK08; LV09; Che+12; SE13b; Liu+16; XWW20; ML20].

4.3.3. Universal update token

This approach treats the time tag as a feature of *MPK* rather than any user's identity and uses it in such a way that all users can receive the same update token to update their individual keys locally.

Approaches belong into this class if, in every epoch, the users are supplied with an update token for their key material that they have to apply to their old key material and if this update is identical for all non-revoked users in the system, such that even a revoked user could update their old key with it.

The class is characterized as follows:

- The updated *usk* depends on the old one.
- The universal update token must not be published, because the owner of a revoked key (or an attacker who holds it) could update the revoked key in that case. However, it is possible to take advantage of tree-like distribution structures, since the information is the same for all non-revoked users. The distribution complexity is therefore logarithmic.
- The computation complexity is constant.

- Since user secret keys depend on *MPK*, is must always be updated. The updated *MPK* can be published if computing the update token from the new *MPK* is hard (as is the case in [Gug20]).
- Besides *msk* and *usk*, the update token is an additional attack vector. It is even more interesting for an attacker than in the case of individual updates, because it can be used to update *any* recent decryption key the attacker holds rather than a specific one. Computing a previous *usk* is, again, possible if the update mechanism is vulnerable to decryption key exposure.

[Gug20] is an example for this class.

4.4. Assessment summary

This chapter answers **RQ 1)**: Revocation mechanisms in IBC are systematically similar because only implicit, indirect means of revocation are consistent with the underlying idea that public keys (and by extension, their validity) should not have to be looked up. This allows an abstract, generalized formalization of identity-based revocation mechanisms for automated analysis (see Chapter 5).

At the same time, the mechanisms differ systematically with respect to their update logic, and three distinct classes can be recognized in the literature.

For a concrete automated analysis, it is meaningful to consider and compare at least those three classes. Individual-token update mechanisms can additionally be separated into mechanisms with re-randomization and those without, so Chapter 6 describes four distinct models instead of three.

5. Formal model of revocation in Identity-based cryptography

This chapter formalizes IBC-revocation to fit the logic of a trace-based automated reasoning tool:

- Section 5.1 gives an overview of the three building blocks in a trace-based model:
  - instructions to model the possible protocol states
  - a formalization of the security goals
  - a formalization of the cryptographic dependencies and adversary knowledge (which are intertwined)
- Section 5.2 provides a minimal communication scenario with all important events that the formalization should be able to capture.
- Section 5.3 derives a state diagram fitting all three classes from the minimal example and formalizes each state. It explains the execution policies assumed in the model and the abstraction of secure channels between TTP and each user.
- Section 5.4 formally states the security properties that a revocation mechanism ideally fulfills, derived from literature.
- Section 5.5 describes the information the adversary can compromise and the mathematical dependencies that the formalization should capture. Together, they model the adversary knowledge.

The formalization allows a clear comparison between the three classes that are identified in Section 4.3: The trace-construction guidelines and the security notions remain the same for all three, but the adversary's options for attacking the respective mechanisms can be tailored to each of them individually.

5.1. Overview

A trace-based reasoning tool requires the following information for any model of a cryptographic mechanism:

Trace construction guidelines describe the protocol so that the prover can reason about all possible protocol runs. The guidelines determine

- the order in which states appear on the trace;

- possible restrictions from execution policies: For example, if each user may receive only one secret key from the TTP, but this policy is not implemented by the protocol logic and instead handled through the TTP's key management policy, the trace construction guidelines may specify this.

Desired security properties are formulated as statements about all possible traces. They either specify that certain undesired behavior occurs on none of them or that reaching a certain state implies the occurrence of another certain state on the same trace. For example, an automated reasoning tool may prove to a user that if neither *msk* nor any of her keys are ever compromised, an attacker can not decrypt messages intended for her (by any means captured in the model).

Attacker knowledge and abilities There are two ways for an attacker to learn information:

- Directly compromising an agent: The model should specify the information an adversary can get from compromising any entity and learning their secret values directly.
- Deriving it from other information: The mathematical interdependence between keys or cryptographic computations allows an attacker to learn new values from known ones. The level of detail with which these dependencies are modeled determine the weaknesses the analysis can or can not find.

5.2. Minimal revocation example

For a clear idea which events the model needs to cover, one must first identify the various settings that can occur. Meaningful two-party communication within a group should allow

a) adding new, legitimate users to the group in any epoch and providing them with valid key material for each epoch in which they are part of the group.

b) encrypted (authenticated) communication between any two non-revoked users in the group in any epoch.

c) removing a user from the group such that she can not decrypt (sign) messages in the following epoch(s), but the remaining group members can.

d) re-admitting users in such a way that they can not decrypt messages from or sign messages for epochs in which they were not part of the group.

Functionality b) implies: Whether or not two legitimate group members can communicate must not depend on their history in the group or how many epochs ago either of them joined. It must also not depend on any group event that affects neither of them (e.g. other users joining or leaving the group, or being expelled for any epoch).

A suitable model correctly captures at least the following events in this order (see Figure 5.1):

(I) The TTP sets up a new group (including master key material) and admits at least two users, User A and User B, in the first epoch. User A can send an encrypted (signed) message to User B, which User B correctly decrypts (verifies).

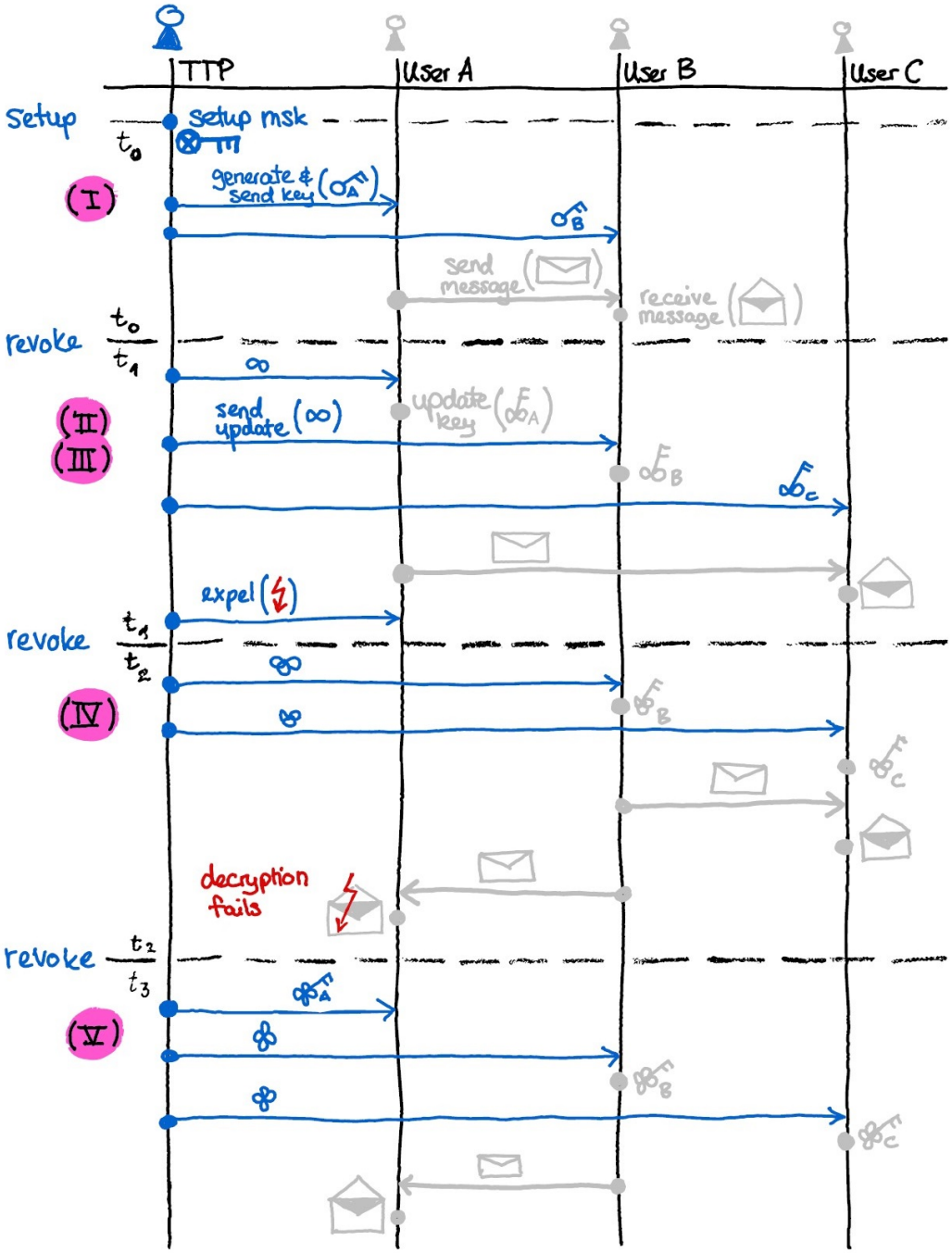

Figure 5.1.: Example run of a group communication that contains user admittance after the first epoch transition, expelling and, in a later epoch, re-admitting a user, updating remaining users and exemplary communication between the different users in different epochs

(II) The TTP revokes all keys and starts a new epoch, but does not expel any users. All users receive an update to their keys and can continue to encrypt (sign) and decrypt (verify) messages correctly.

(III) The TTP adds a user C to the group after the first epoch. User C can correctly decrypt (sign) a message from (to) another group member in the current epoch.

(IV) The TTP expels a user, e.g. User A, from the group. All user keys from that epoch are immediately revoked. User B and User C receive key updates for the following epoch and can continue to communicate. User A can not correctly decrypt (sign) a message from (in) the following epoch.

(V) After User A's removal, the TTP adds a new User D to the group who can correctly communicate with User B and User C.

Figure 5.1 illustrates a minimal example that contains events (I) through (V) (indicated in **pink**). On the horizontal axis at the top, it shows the four active parties in the communication example: The TTP ( ) and three group members User A, User B and User C (, ,). A vertical line (*"timeline"*) for each party shows the chronological order of group management- and communication events (where only one action at a time can happen) from start (top) to end (bottom).

The minimal example spans over four epochs t_0-t_3. The beginning of each epoch is demarcated with a coarsely dashed pencil line ( ).

The first epoch starts with the setup of the TTP, who chooses an *msk* and publishes the corresponding *MPK*. Each following epoch is initiated by the TTP who revokes all user keys.

In each epoch there are **TTP-actions (denoted in blue)** and **user actions (denoted in gray)**. A blue or gray dot on a timeline denotes a point in time at which the corresponding party takes an action. A blue or gray arrow may point from the dot to the timeline of a user who is directly affected by this action. (The TTP is never affected.) Figure 5.1 illustrates the following scenario:

Epoch t_0: Event (I) The TTP sets up a communication group by choosing an *msk* ( ). This starts epoch t_0 (**setup**). Then it generates a *usk* and sends it to User A ( ), thereby adding User A to the communication group. In the same way, it adds User B. User A encrypts a message for User B and sends it out ( ). User B successfully decrypts the message ( ). Then, epoch t_0 ends, which automatically revokes all keys (**revoke**). This process is managed by the TTP, who determines the epochs.

Epoch t_1: Events (II) and (III) Epoch t_1 starts. Neither user is expelled, so both receive an update ( ). The kind of update (new keys, individual update, or universal update) depends on the encryption (signature) scheme that is used. Both users apply their update to the key they already have if the scheme is designed accordingly ( ). Figure 5.1 illustrates a universal update (a two-ring, three-ring, or four-ring token sent to each remaining group member). Independent of the update, User C is added to the group. User A encrypts a message and sends it to User C, who successfully decrypts it. Then, User A is expelled by the TTP ( ). Epoch t_1 ends, which implicitly revokes all keys.

Epoch t_2: Event (IV) At the beginning of epoch t_2, User B and User C receive an update. User A does not receive an update, having been expelled at the end of the previous

epoch. The users apply their updates. Then, User B encrypts and sends a message to User C, who successfully decrypts the message. User A also receives an encrypted message from User B, but fails to decrypt it ()[1]. Epoch t_2 ends, which implicitly revokes all keys.

Epoch t_3: Event (V) Epoch t_3 starts. User A is re-admitted to the group (in lieu of a completely new user) and receives a new key from the TTP. User B and User C receive updates for their respective keys again and apply them. User B sends an encrypted message to User A, who successfully decrypts it.

Figure 5.1 deviates from the previously described events in the following ways to be more succinct:

1. The difference between admitting an entirely new user and re-admitting User A is abstracted. It treats User A as a new user, who receives fresh key material rather than an update for their old key material. An ideal model covers both re-admittance of User A and adding new users after one or more users have been revoked.

2. The effect of revocation on decrypting a message is illustrated, but not the effect on signing. For the rest of the chapter, encryption is assumed as the default because the case of signatures can be seamlessly inferred from the case of encryption.

The minimal example shows a protocol run containing all five core events in four epochs. However, the automated prover should ideally not be limited to proving security only for this exemplary run but allow arbitrarily long protocol runs where any of these events occur arbitrarily often in any (meaningful) order between arbitrarily many users. Note that effects between two or more communication groups are outside the scope; the model is limited to one communication group.

5.3. State diagram and formalization of each state

The basis of the protocol analysis are "traces", protocol runs inferred by the program. They reflect the protocol agents' possible states and the order in which agents can transition between them.

To model any protocol, it is therefore essential to have a clear understanding of its state diagram.

There are four main phases in every epoch of the communication group

- `SETUP`, where the master parameters are determined
- `GET`, in which a user gets a valid secret key for the current epoch
- `USE`, in which users encrypt and decrypt messages or sign and verify them
- `DELETE`, in which all epoch-bound values become obsolete and all user secrets are revoked

[1] Note: User B does not necessarily know whether User A is still a member of the group or not. Not having to check User A's status is an express advantage of IBC, but it comes at the cost of unnecessarily encrypting messages for which the receiver has no key. Attacker models in which such messages are attack vectors need to consider this. The presented attacker model omits such attacks.

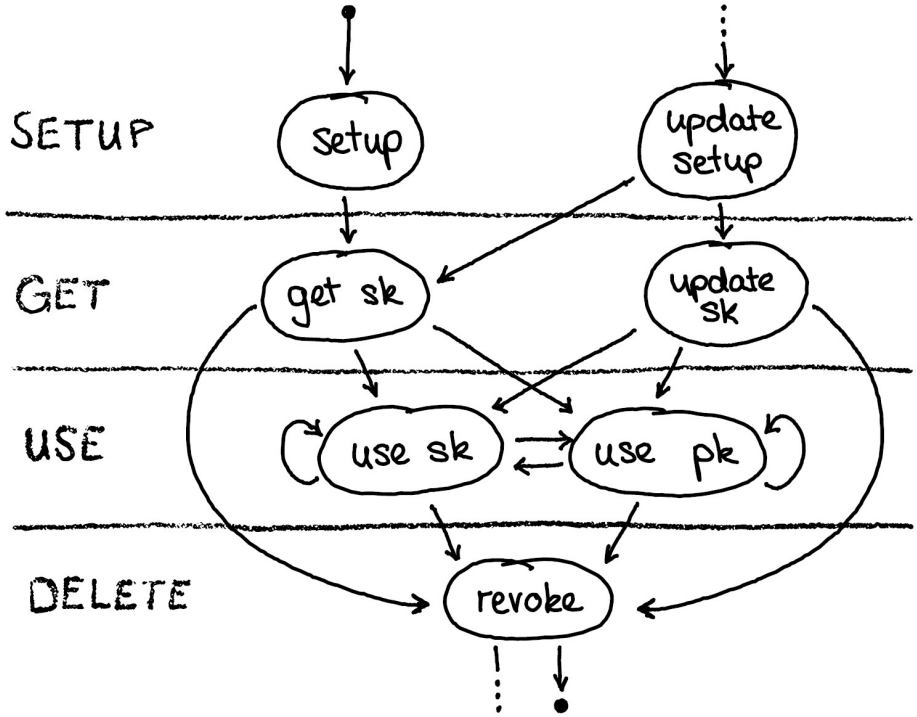

Figure 5.2.: State diagram of identity-based revocation

Figure 5.2 shows the flow of the protocol with the following states and the order in which they can occur (denoted by black arrows). Every state belongs to one of the four main phases, which are indicated on the left and separated by horizontal lines.

setup: The TTP chooses a secret key msk_1 and computes MPK_1 from scratch. This is the first state entered in any instance of the process (as denoted by the arrow descending from a black dot) and the beginning of the first epoch. MPK_1 and the first time stamp are published during the setup. The setup state only occurs once.

get sk: A user receives her first secret key (sk) from the TTP after (re-)entering the group. This state can only be entered after the first *setup* or any *update setup* state of the TTP (denoted by arrows pointing from these states to *get sk*). To generate this key, the TTP needs msk_t for the current epoch t, an identity ID of the user and the current time stamp t. It outputs a corresponding $usk_{ID,t}$ and sends it to the user over a secure channel (which is abstracted in this formalization and assumed to be perfectly secure: authentic, confidential, integrity-protected etc.).

use pk: Users may use another user's identity (= their public key pk), MPK and a current time stamp t to encrypt a message or verify a signature. Even those who do not have a user secret key themselves can use the public values of others. If MPK is epoch-independent, users can encrypt messages for future epochs or non-existent users. This can be modeled if future time stamps or identities are predictable. For simplicity, we assume that this state can only be reached once the corresponding secret values exist; therefore, this state can only be reached after the user received valid key material for through *get sk* or *update sk*.

use sk: A user ID decrypts or signs a message with her own $usk_{ID,t}$ that corresponds to the epoch t for which the message is or was intended. To decrypt, a user needs the corresponding $usk_{ID,t}$ and the ciphertext. To sign, she needs $usk_{ID,t}$ and a message. This state can be reached once the user has a secret key either from entering the group (*get sk*) or from updating an old key (*update sk*).

revoke (advance epoch): The epoch changes and all previously valid keys become obsolete (= are revoked). A new (unique) time stamp is generated or computed from the previous time stamp and published. It is determined which users are expelled from the group, i.e. which users should not receive a re-key or an update token. If the time stamps are not generated predictably, it is important to record the succession of time stamps so that the model allows checking dependencies, e.g. whether $usk_{ID,t-1}$ is from the epoch directly preceding the current epoch t, for example. This state can (sensibly) be reached at any time as soon as there exists at least one $usk_{ID,t}$ to revoke.

update setup: If the mechanism requires it, msk_{t-1} is updated to msk_t and MPK_t is derived and published for the new epoch. As indicated by the dotted arrow pointing to this state matching the dotted arrow line out of *revoke*, *update setup* always follows the *revoke*-state unless the communication group is deleted for good (denoted by the arrow pointing to the black dot).

update sk: The remaining users receive a re-key or an update for their secret key material. For this state, users need either a new key or an update token from the TTP. Like before, the TTP sends these values to the user over a secure channel (which is abstracted in this formalization and assumed to be perfect). This state can be reached for every user as soon as the TTP has advanced the system to the next epoch and updated the setup values, if necessary.

The state diagram is the same for renewal-, individual-update-based or universal-update-based revocation. The differences lie in the specific handling of keys in each case (see Section 5.5). For some approaches, some of the actions do not change the system state (e.g., if the setup values are never changed) and could be left out of the diagram; but the verbose version of the diagram still applies.

5.4. Security properties

The automated analysis needs to capture established, intuitive security notions for the revocation mechanisms on a level of abstraction consistent with the protocol abstractions.

The following sections describe

1. Forward Security
2. Post-Compromise Security
3. Decryption Key Exposure Forward and -Backward, and
4. Collusion Resistance

Notions 1, 2, and 3 are illustrated in Figure 5.3.

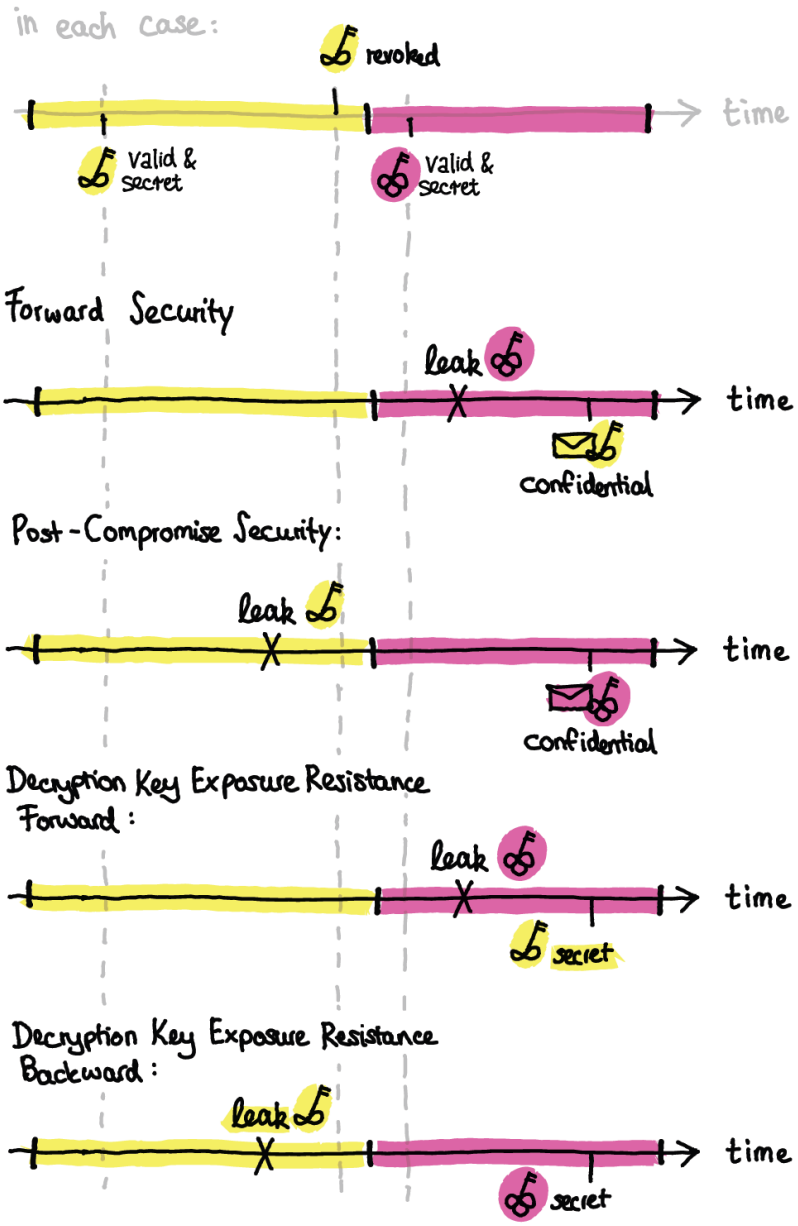

Figure 5.3.: Each security property concerns the leak of a user secret key at a certain time and what it implies for the security of a message/user secret key from another epoch. The topmost, gray time axis ($\longrightarrow time$) indicates that in each of the illustrated cases, a *usk* is generated (**yellow**) at the beginning of the epoch and revoked when the epoch ends. The system advances to the next epoch (**pink**) and a new *usk* is generated (from scratch or through an update). The revocation of this key is not shown (it is irrelevant). The other four, black time axes ($\longrightarrow time$) illustrate four distinct security notions, each indicating the leak of a certain key in one of the two epochs and which messages or keys remain secure if the security goal is achieved.

Note that all stated properties are commonly found in related literature, albeit not in this specific combination and without a distinction between the consequences of decryption key exposure for previous and for future epochs. They only relate to encryption and confidentiality. For signatures and authenticity, equivalent properties apply[2].

Notation Below, the following notation is used:

- If t denotes an epoch, then $t \in \mathbb{N}$ is assumed and the arithmetic symbols are used intuitively in $\mathbb{N}$; for example, $t + x$ denotes the x^{th} epoch after t.
- $usk_{ID,t}$ is the user secret key that is valid for an identity ID in epoch t.
- msk_t is the master secret key that is valid in epoch t. If the master secret key is never updated, the notation still applies and msk_t is the same for all epochs t.

5.4.1. Forward Security

This property is in the standard repertoire of security notions and defined for authentication and both symmetric and asymmetric encryption mechanisms. The term "Forward Secrecy" is more commonly used for key exchange mechanisms, though not exclusively [BG21]. For asymmetric encryption, Forward Security means that messages encrypted with a public key from an epoch t remain confidential even if a secret key of the same user from an epoch $t + x > t$ is compromised (but not the secret key from epoch t itself). This concept is easily transferable to IBC. In this work, Forward Security therefore means that compromising $usk_{ID,t+x}$ from an epoch $t + x > t$ does not give an attacker any advantage when trying to decrypt a message that was encrypted for an identity ID and an epoch t. Only traces in which the leak of $usk_{ID,t+x}$ is part of the adversary's attack are considered proof that Forward Security does not hold or only conditionally holds in the modeled approach.

5.4.2. Post-Compromise Security

This property was first introduced in the context of authenticated key exchange, i.e. protocols to establish a shared key between two parties who authenticate each other in the process [CCG16]. Informally speaking, a mechanism achieves Post-Compromise Security if the communication between two users remains secure even if one party's keys have been compromised. Applied to IBC-revocation and encryption, Post-Compromise Security is achieved if compromising $usk_{ID,t-x}$ from an epoch $t - x < t$ gives an attacker no advantage when trying to decrypt a message that was encrypted for an identity ID and an epoch t. As before, only traces in which the leak of $usk_{ID,t-x}$ is part of the adversary's attack are considered proof that Post-Compromise Security does not hold or only conditionally holds in the modeled approach.

5.4.3. Decryption Key Exposure Resistance Forward and -Backward

Decryption Key Exposure Resistance is introduced as a security notion in [SE13b] to account for a weakness of individual update mechanisms in which a part of the key changes in each epoch but another part remains fixed.

[2]See, for example, [BG21] for an explanation of forward secure signatures.

In these cases, an adversary can pair the static part of the key with any update token to create a valid key. To understand the security of a RIBE mechanism, three kinds of leaks need to be considered in the adversary model: of the static key, of the update token, and of the decryption key derived from them). However, previous models only considered a leak of either key part, not of the full decryption key.

The security model in [SE13b] contains all three kinds of leaks and allows decryption key exposure at any time (except the target epoch). Since the static key is re-randomized with a new token in every epoch, there is no "chain of updates" in individual token approaches; instead, the updated keys share a direct ancestor (= the static key).

Unlike the security model in [Gug20], the proposed notion does not distinguish between effects from decryption key leakage before or after an update, even though doing so is straightforward.

This work therefore differentiates between Decryption Key Exposure Resistance Forward and -Backward:

Decryption Key Exposure Resistance Forward A mechanism achieves Decryption Key Exposure Resistance Forward if the adversary can not learn $usk_{ID,t}$ without compromising the key itself or msk_t, even if $usk_{ID,t+x}$ for any later epoch $t + x$ is compromised.

Decryption Key Exposure Resistance Backward A mechanism achieves Decryption Key Exposure Resistance Backward if the adversary can not learn $usk_{ID,t}$ of an epoch t without compromising the key itself or msk_t, even if $usk_{ID,t-x}$ from a previous epoch $t - x$ is compromised.

As before, only traces in which the leak of $usk_{ID,t+x}$ ($usk_{ID,t-x}$) is part of the adversary's attack are considered proof that Decryption Key Exposure Resistance Forward (Backward) does not hold or only conditionally holds in the modeled approach.

Note that the security goals of Decryption Key Exposure Resistance on the one hand and of Forward- and Post-Compromise Security on the other are similar but not identical: While Decryption Key Exposure Resistance aims at whether an attacker can obtain a key, Foward- and Post-Compromise Security require that an attacker does not break the confidentiality of a message. Achieving the former for one direction (forward or backward) implies achieving Forward/Post-Compromise Security, respectively, because obtaining a key allows decrypting a message.

5.4.4. Collusion Resistance

Collusion Resistance is an important notion for IBC-mechanisms in which a user identity consists of distinct components ("attributes") and all users who share an attribute know the associated share of the secret key. In these cases, users with distinct attributes may collude to derive a key that none of them holds individually and that corresponds to an identity that is a mix of their respective attributes [SW05].

In this work, each identity is associated with only one key, not a set of keys, so the notion does not directly apply. However, it is useful in the following sense (as, for example, used in [BGK08; SE13b]): An update mechanism is called collusion resistant if it does not allow any number of users to decrypt a message together that none of them should be able to decrypt on their own.

For the prover to reason about security notions, a model must reflect structural and mathematical dependencies between keys of various epochs and various users. Together with the information the attacker can compromise, these elements yield the attacker model described in the next section.

5.5. Adversary model

To understand whether a specified mechanism achieves a specified goal under realistic circumstances, the model must specify the adversaries against which the mechanism should be tested.

For the best security guarantees, the adversary should be restricted as little as possible: The stronger the adversary, the stronger the security of a mechanism that withstands it. The following aspects shape the adversary model:

- The private channel from the TTP to each individual user is assumed to be perfect, and thus the mechanisms securing the *usk* distribution are not modeled. Other than that, all channels are assumed to be untrusted, so the adversary knows all public values (like the *MPK* and the group members' identities) and can receive, modify, and intercept any messages in the network and interject new ones.
- The adversary can appear as a legitimate group member and register (= receive *usk*s for) as many identities as it chooses.
- The user management, which is done by the TTP, is abstracted and assumed to be ideal: The adversary can therefore
  - not register an already registered identity with the TTP,
  - not advance epochs independently from the TTP
  - not get the TTP to ignore user- and key management policies (e.g. that fresh values are used only for new or re-joining members)
  - not manipulate the TTP in any other way than compromising and then using its secret values.

To understand how the adversary might break the security, i.e. what information the adversary can derive from which inputs into a cryptographic protocol or mechanism, the possible sources of adversary knowledge need to be modeled. There are two ways for the adversary to obtain a secret value:

1. She directly receives this information (for example by compromising a user and stealing one of their secret keys). For an exhaustive model, the adversary should be able to compromise any public and private information as soon as it exists or at any time thereafter. However, for non-trivial insights, the adversary may not compromise *all* entities. Otherwise, there is nothing to show.
2. She derives it by exploiting the structural, mathematical or cryptographic dependencies of other known values

The next two sections detail these methods.

5.5.1. Adversary knowledge

The model must let the adversary know all public values:

- the current epoch t and all previous epochs
- MPK of the current epoch t
- the identities of all registered users (which are their public keys)
- all transmitted cipher texts (since they are sent over insecure channels)
- all public update values for all epochs and identities once they exist

It must also allow the adversary to access secret values on demand. It is important to include all values that are generated or computed in the process and could possibly be leaked: For example, a failure to model "decryption key exposure" (what in this model is called usk leak) in the proof for [BGK08] made it impossible to detect an attack.

The following values need to be included if they exist in an approach:

- msk_t for any epoch t
- $usk_{ID,t}$ for any identity ID and epoch t in which this key exists
- secret update values for any epoch and identity in which they exist
- static user secret keys $susk_{ID,t}$ for any identity ID and any epoch t in which they exist

It is also public knowledge how keys and update values are computed. The next section details the dependencies for the model to capture.

5.5.2. Cryptographic primitives and mathematical dependencies

To prove an adversarie's inability to exploit mathematical connections between keys, cipher-texts or other cryptographic information, the dependencies in question should be modeled as precisely as possible. However, as has been shown in [BHS19][3], most mathematics underlying IBC-schemes rely on distributive laws and are therefore hard to model in trace-based tools. The essential logic of both identity-based signatures and identity-based encryption can be and has been modeled, and the goal of this work is to model the essential logic of revocation and update mechanisms in IBC.

For a meaningful model of update- and re-key-based revocation mechanisms, the corresponding mathematics need to be captured abstractly, between keys and cipher texts (signatures) as well as between various keys[4] [5].

[3] The paper focuses on TAMARIN, but other tools like ProVerif have similar limitations regarding distributive laws.

[4] Dependencies between cipher texts (signatures) of different epochs are not included, because this work is strictly concerned with updatable keys, not with updatable or malleable encryption (signing), which is a different area of research (see for example [LT18]).

[5] While the characteristics described below are standard and appear in all considered schemes, they must be double-checked when applying the formalization for a new scheme.

between msk_t and MPK_t In all considered schemes, MPK is derived from msk in a way that is hard to reverse (e.g., based on the discrete logarithm problem). Therefore, derivation of MPK_t from msk_t must be modeled, but the reverse must not be allowed.

between ID, msk_t, t and $usk_{ID,t}$ The user secret depends at least on the user identity and msk_t, and may also depend on the epoch identifier t. In all considered cases, computing msk_t from the user secret is hard.

between the keys, the plaintext and the ciphertext The model must allow anyone who knows the public values to encrypt a plain text for a certain identity. It must allow anyone to decrypt the resulting ciphertext if, and only if, they have the corresponding user secret key. The model needs to specify what constitutes this "correspondence", namely that the identity, the underlying msk and other values binding the user secret to the current epoch need to match in a specific way.

between msk_t and msk_{t+1} If msk_t and msk_{t+1} are not the same, the model needs to capture whether computing one from the other in either direction is possible or not. Note that MPK_t will change, too, if msk_t changes, but that this is captured by the derivation rule between msk_t and MPK_t.

between $usk_{ID,t}$ and $usk_{ID,t+1}$ The model must capture whether and, if so, how user keys can be derived from each other in either direction of time and between identities.

between epoch identifiers If epoch identifiers (here: t, $t+1$) have an underlying mathematical logic, this might interfere with other dependencies and needs to be be modeled.

These dependencies determine the differences between the three classes of IBC-revocation mechanisms considered in this work. For example, the dependency between $usk_{ID,t}$ and $usk_{ID,t+1}$ is entirely non-mathematical in the key renewal approach and thus, in the model, whereas the dependency for universal update tokens must model that an adversary can use the token to update the key for any identity.

The model can therefore be tailored to each class while the main construction guidelines and the security properties remain the same. This allows a clean comparison between the mechanisms. The model assumes that every message is unique. Dependence between plain texts is out of scope for this work.

5.6. Formalization summary

The formalization consists of three parts: 1. Trace construction guidelines, 2. desired security properties and 3. adversary knowledge.

The following overview summarizes the information that needs to be modeled for a trace-baced automated analysis of revocation mechanisms in IBC as laid out above:

Trace construction info	Security Properties	Adversary Knowledge
necessary states: • setup • get sk • use pk • use sk • revoke • update setup • update sk	• Forward Security • Post-Compromise Security • Forward Decryption Key Exposure Resistance • Backward Decryption Key Exposure Resistance • Collusion Resistance	• public information • direct compromise options: – $\forall t : msk_t$ – $\forall ID, t : usk_{ID,t}$ – $\forall ID, t : susk_{ID,t}$ – private update values • mathematical dependencies between – msk_t and MPK_t – msk_t, ID, t and $usk_{ID,t}$ – msk_t and mst_{t+1} – $usk_{ID,t}$ and $usk_{ID,t+1}$ – keys, plain- and cipher texts – t and $t + 1$

This chapter answers **RQ 2)** ("How can revocation approaches and their security requirements be formally modeled such that a trace-based automated prover can reason about them?"). Chapter 6 introduces the translation of this formalization to Tamarin. As explained before, the trace construction information and the security properties are modeled once for all classes, as are the compromise options. The differences between the classes are reflected in how the mathematical dependencies are implemented and in the information that is public or private. Chapter 7 reports and interprets the results of executing the model, and contains a sanity check that the minimal example brought forward in this chapter can be modeled with the proposed implementation.

6. Modeling IBC-revocation mechanisms

As a proof-of-concept, this chapter shows that the formalization presented in Chapter 5 can be meaningfully translated for an automated prover.

- Section 6.1 explains why Tamarin is the prover of choice for this work.
- Section 6.2 introduces Tamarin's logic and the modeling options.
- Section 6.3 through Section 6.5 detail the translation of the formalization to a meaningful Tamarin model. The translation is presented along the same lines as in Chapter 5:
  - in Section 6.3, the state diagram
  - in Section 6.4, the security properties (which are the same for all classes)
  - in Section 6.5, the adversary knowledge (which reflects the differences between the classes)
- Section 6.6 describes some sanity checks that the model includes to ensure that it represents the formalization coherently.

Appendix A contains a code blueprint which is the basis for all models. Placeholders (highlighted in yellow) indicate where approach-specific code must be added to get a functional and faithful model. The explanations in this chapter reference the blueprint and the placeholder insertions in the corresponding model codes (Appendix B through Appendix E).

6.1. Tool choice

The prover of choice for this work is Tamarin because

- it proves correctness (other than, for example, Verifpal [KNT20], which can find mistakes but not prove their absence),
- proves properties for "unbounded sessions", i.e., regardless of the number of sessions running in parallel (other than, for example, AKiSs [Cha+16] and DeepSec [CKR18]), and
- has a more precise Diffie-Hellman model than ProVerif [Bla16] and includes bilinear pairings.

Also, Tamarin was the prover of choice in a previous publication [Gaz+21b] and thus a familiar tool. In hindsight, the Diffie-Hellman model could not be used as anticipated in this work: Therefore, Tamarin and ProVerif are equally well-suited.

6.2. Tool background

TAMARIN is a *trace-based* automatic prover that works with *multi-set rewriting* [Bas+17].

A multiset is a set in which elements may appear more than once. The prover starts at a given *(multi-)set* of facts representing the state of a system at a given time. It *re-writes* this set, backtracking an origin for each fact, and thereby constructs a *trace* that represents a possible (partial) protocol run.

The multi-set rewriting system is expressed with the following building blocks:

1. Atomic terms, function symbols and the equational theory (which assigns meaning to function symbols)
2. Facts, the atomic elements of a state
3. Rules and action facts for constructing and reasoning about traces and to assign meaning to function symbols
4. Restrictions on considered traces
5. Lemmas to reason about trace properties

These elements are explained below. For further details consult the TAMARIN manual[1].

6.2.1. Atomic terms, functions and the equational theory

There are various sorts of atomic terms, each indicated by a typographic marker. Most notably for the rules in Section 6.3 there are

- fresh variables like ~y
- public variables like $X (that are often used to identify and distinguish protocol agents)
- global constants like 'c'

New terms can be created from existing terms using function symbols. Equations give the function symbols meaning. Meaning can also be assigned by how symbols interact in rules. Some functions and equational theories are built-in and can seamlessly be used in models, for example hash functions, public key encryption, or Diffie-Hellman exponentiation.

As an example, consider public key encryption: With the following line in a rule, the fresh value ~sk can be identified as the secret key for a public key (using the unary built-in function pk):

```
pub_key = pk(~sk)
```

The equation for public key encryption describes how the binary functions for asymmetric encryption enc and decryption dec are interrelated:

```
dec(enc(m, pk(sk)), sk) = m
```

[1] version published on November 10, 2022; see https://tamarin-prover.com/manual/index.html for the current version and https://github.com/tamarin-prover/manual-pandoc for github version

This specifies that function `dec` yields a value `m` (the message) if a) its first argument matches the encryption of `m` with a public key for `sk`, and b) the second argument is `sk`, matching the public key used for encryption.

A new term representing the ciphertext for m can therefore be defined as

```
cipher= enc(~m, pub_key)
```

6.2.2. Facts

Facts are state descriptions that can take terms as inputs. In the model, three notations appear:

- `Fr(~x)` denotes the input of a *fresh* variable, for which a new value is assigned every time the rule is applied and which needs no further backtracking. The rule can use the fresh value `~x` in the output facts.
- `In(x)` and `Out(x)` denote information received and sent over a public channel. The public channel represents a network that is under the adversary's control. The adversary reads all information `x` that is sent over the public network (`Out(x)`) and can generate `In()`-facts from values it knows.
- Custom-defined facts `SomeFact(~y, $X, 'c')` can bind values of different sorts together. Each instance of a fact originates in the output of a rule. It is consumed when a rule uses it as input. Therefore, each instance can only be used once.

  To use them arbitrarily often, custom-defined facts can be made *persistent* with a prefix "!". Once a persistent fact `!SomePersistantFact(~y, $X, 'c')` is output by a rule, it remains in the multi-set indefinitely (unless restrictions apply).

6.2.3. Rules and Action Facts

Rules represent state transitions. They take facts as input and produce facts as output. They are denoted as follows:

```
rule some_transition:
[ (inputs) ]
-->
[ (outputs) ]
```

To reason about the protocol or restrict the traces for Tamarin to consider, rules can be labeled with *action facts*. Consider the following example where the rule is labeled with the action fact "OnlyOnce()":

```
rule generate_master_keys:
[ Fr(~sk) ]
--[ OnlyOnce() ]->
[ !MSK(~sk)
, !MPK(pk(~sk)) ]
```

The rule is called `generate_master_keys` and takes a fresh value ~ `sk` as input. It outputs a persistent fact `!MSK(~ sk)` to record that the value ~ `sk` can be used as a master secret key. Another persistent fact `!MPK(pk(~ sk))` records that `pk(~ sk)` can be used as the master public key corresponding to ~ `sk`.

Section 6.2.4 provides an example for using this action fact to restrict a model.

6.2.4. Restrictions

Restrictions determine the set of traces that Tamarin includes in the proofs. They are formulated in first-order logic (FOL) over action facts and time points (a special type of variable, prefixed with #, that only appears in FOL-formulas). Every rule execution happens at a unique time point.

To ensure, for example, that a rule is executed at most once, label it with the action fact `OnlyOnce()` and include the following restriction in the model:

```
restriction OnlyOnce:
"All #i #j. OnlyOnce()@ i & OnlyOnce()@ j ==> #i = #j"
```

It says that for all time points `#i` and `#j` on the trace it must hold that: If the action fact `OnlyOnce()` appears both at `#i` and at `#j` (that means, any rule with this action fact is executed at time points `#i` and `#j`), then the time points must be the same.

Given this restriction, the prover only considers traces for the model in which this formula holds. Traces with more than one appearance of the label `OnlyOnce()` are not considered for proofs.

6.2.5. Lemmas

Lemmas are statements about the model. There are two classes:

- "All"-statements: If Tamarin proves this, all protocol runs with the specified properties fulfill the lemma. Disproving this means that Tamarin can construct at least one protocol run for which the lemma is not true.
- "Existence"-statements: If Tamarin proves this, at least one run of the protocol fulfills the lemma. Disproving this means that Tamarin can not construct any protocol run to fulfill the lemma.

Not all lemmas can be proven or disproven: Tamarin may fail to terminate and thus produce no result. If it terminates, the results hold.

The lemmas are analyzed for arbitrary protocol length and for arbitrary many parallel instances of the protocol. An agent may occur in several instances at once and is not bound to have the same role in each of them. (Note that models in this work allow arbitrary many epochs, but the restrictions exclude parallel sessions because they are out of scope.)

In some cases, Tamarin can not deduce all possible sources of a fact. To mitigate this, the model can include "sources-lemmas" that are easier to prove. Tamarin can re-use these statements to reason about other lemmas. *Sources-lemmas* need to be checked for correctness; Tamarin will use them even if they are wrong. As an example, loops and other recurring transitions can be difficult for Tamarin to work with if it fails to identify the common starting point. A *sources-lemma* which makes the starting point for a loop explicit to the prover can help in this case.
The next section explains how the state diagram presented in Section 5.3 is translated to Tamarin with suitable terms and rules.

6.3. The state diagram

All states of the state diagram (except *start* and *end*, which represent start and termination of a proof) are directly translated to rules in Tamarin, which is explained Section 6.3.1.

The restrictions discussed in Section 6.3.2 are necessary to refine the model.

6.3.1. Translating States to Rules

This section explains how each state is translated to a rule or rules:

- State "setup" to `rule setup`
- State "get sk" to `rule generate_usk`
- State "revoke" to `rule advance_epoch`
- State "update_setup" to `rule update_msk`
- State "update sk" to `rule distribute_token` and `rule update_usk`
- State "use pk" to `rule encrypt`
- State "use sk" to `rule decrypt`

Figure 6.1 illustrates the translation, starting at the top:

- Each rule is represented by one box:
  - The pink header displays the rule name (e.g. `rule setup`).
  - The two horizontally separated parts below it represent the inputs (IN/NEW) and outputs (OUT), respectively.
  - The input section is vertically divided into arguments originating from a rule output or from the public knowledge base (IN) and values that are freshly created in each rule execution (NEW).

  Note that the terms "IN" and "OUT" do *not* allude to the `In()`- and `Out()`-facts in Tamarin.
- The bold lines and bold font mirror the states from Figure 5.2, thus making apparent that all states except "update sk" are translated to a single rule, and that "update sk" is divided into the rules `distribute_token` and `update_usk`, so the TTP's and the user's share of this operation are separated.
- Almost all rule names vary slightly compared to the corresponding state in the state diagram to make them more specific (e.g. "encrypt" vs. "use sk") or reflect the process rather than the protocol logic ("advance_epoch" vs "revoke").
- The arrows in Figure 6.1 indicate the input/output connection between rules, i.e. which rule is a possible source (arrow base) of input for the same or another rule (arrow head). Black arrows show input/output connections possible in the first epoch, blue arrows show connections happening after at least one epoch transition. (The dashed arrow lines protruding from `rule advance_epoch` are semantically the same as the blue ones; they are only dashed to make the figure easier to read.)

The following paragraphs describe the rules and the input-/output logic between them.

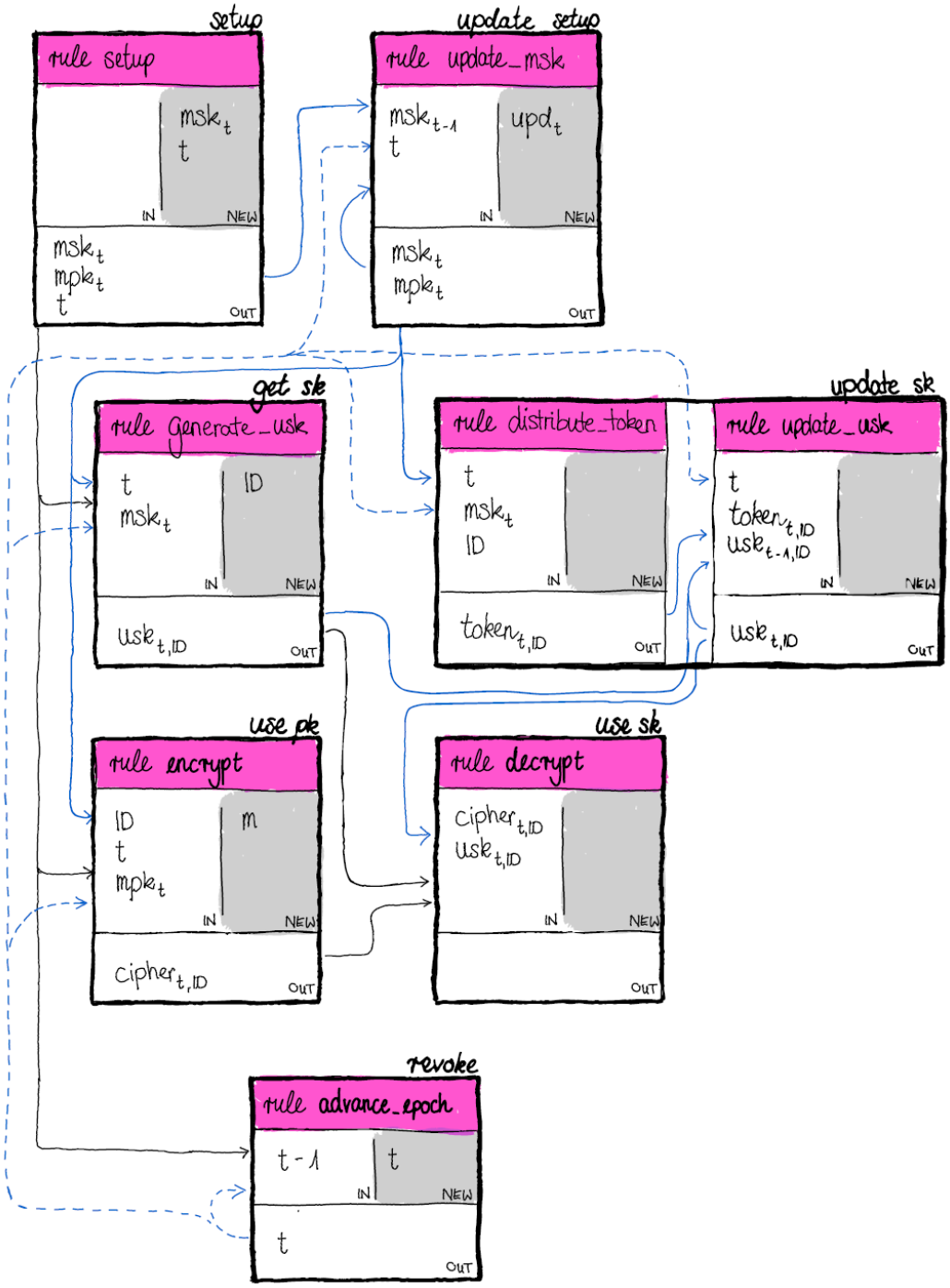

Figure 6.1.: Each state of the formalization (indicated with a black frame and bold label on top) is implemented through one or two rules (boxes with pink headers, labeled with the rule names).
The inputs to each rule are either fresh (gray middle part, labeled NEW) or passed along from other rules (white middle part, labeled IN). The outputs (bottom white part, labeled OUT) are used further, according to the black and blue arrows (black denoting events possible in the first epoch; blue, events possible after the first epoch transition; the output arrows for the epoch identifier `t` from `rule advance_epoch` are dashed for readability).

```
rule setup:
let
%formula for msk_t computation
%formula for mpk_t computation
in
[ Fr(~msk)
, Fr(~t)
]
--[ OnlyOnce(), Is_Epoch(~t),
    Origin_MPK_MSK(~msk, mpk)]->
[ !MSK(~msk, ~t)
, !MPK(mpk, ~t)
, !Epoch(~t)
, Out(mpk)
, Out(~t) ]
```

Listing 6.1: Blueprint code

```
rule setup:
let
msk = ~msk_new
mpk = pk(msk)
in
[ Fr(~msk_new)
, Fr(~t)
]
--[ OnlyOnce(), Is_Epoch(~t),
    Origin_MPK_MSK(msk, mpk)]->
[ !MSK(msk, ~t)
, !MPK(mpk, ~t)
, !Epoch(~t)
, Out(mpk)
, Out(~t) ]
```

Listing 6.2: Code for key renewal

Figure 6.2.: Code comparison for `rule setup` in the blueprint (Listing 6.1) and in the key renewal approach (Listing 6.2)

State "setup" to `rule setup`

This rule (line 10 in Appendix A) represents the start of a protocol run. Therefore, the premise of this rule only requires fresh facts. The starting values for msk_t and MPK_t (written mpk_t in the code) are determined by choosing a fresh value for msk_t and computing mpk_t as `pk(`msk_t`)`. The first epoch identifier is set to a fresh value.

To make values available for other rules, they need to be output by the rule in some form: The mpk_t and t are public values which the attacker may know, so they are published as raw values (= added to the knowledge base via the `Out()`-fact). All values are also made available for other rules through persistent facts.

There are two reasons to model output as a persistent fact:

1. If the value is not also published, it is kept secret from the adversary and can only be used in other rules, but not for the adversary's computations.

2. The values can be re-used as often as necessary and are not consumed at a rule execution (which would happen with linear facts). This is, for example, necessary so the same msk_t can be used to generate several user keys. Executing `rule setup` several times to obtain a linear fact for each usk-generation would generate a fresh msk_t each time instead of the same; so linear facts are not helpful here.

Three rules may use these persistent facts in the first epoch (illustrated by the black arrow starting at `rule setup`): The msk_t is necessary to generate user keys, the MPK_t is needed to encrypt messages, and the epoch identifier t is needed for key generation, encryption and to transition to the next epoch. If the TTP's values are time-invariant and the state "update setup" is not modeled explicitly, the outputs are also used in later epochs and for user key updates (not illustrated). Starting with the second epoch, `rule update_setup` may use msk_t to update it for the new epoch (illustrated by the blue arrow starting at `rule setup`).

The mathematical connections between the keys are defined individually for each approach; see Section 6.5 for details and discussion on all the placeholders appearing in this and the other rules.

State "get sk" to `rule generate_usk`

This rule (line 25 in Appendix A) is called "generate_usk" instead of "get sk" because it encompasses both the generation and the (implicit) distribution of the user secret key, not only the state in which a user gets their key.

The premise requires an epoch identifier and msk_t as persistent facts, so they are not consumed and can not be supplied by the adversary but must come from another rule execution. With an approach-specific computation, the rule generates $usk_{ID,t}$ for an identity ID from msk_t. The ID is specified as a public value, meaning the prover uses a (fresh or existing) public value for it when executing the rule. In the conclusion, $usk_{ID,t}$ is output as a persistent fact that binds it to the corresponding epoch and identity.

State "revoke" to `rule advance_epoch`

This rule (line 35 in Appendix A) describes the implementation rather than the logic of the protocol, so it is called "advance epoch" rather than "revoke". Since keys are revoked indirectly by transitioning the whole system to the next epoch, this accomplishes the revocation.

Given the current epoch through a persistent fact from either the first setup or from a previous execution of the rule itself, the TTP chooses a fresh epoch identifier.

The premise requires an epoch identifier as a persistent fact and supplies a fresh value. The conclusion outputs the new value as a new epoch identifier, both by publishing it and as a persistent fact, and outputs a reusable fact to record the order of the two epochs for the update-rules. The rule has two labels, `EpochEnded(~t1)` and `Consecutive(~t1, ~t2)` that are used in restrictions and lemmas to account for the correct use and order of epochs.

This rule is independent of the chosen revocation approach.

State "update_setup" to `rule update_msk`

This rule (line 45 in Appendix A) requires an epoch identifier t_{new}, the master secret from the previous epoch (together with a fact that records the order of old and new epoch), and a fresh update value in the premise. From the previous msk_{old} and the update value, the secret key msk_{new} for this epoch is computed according to the approach's logic. Usually, mpk_{new} is connected to msk_{new} via the public key function `pk()`, so this assignment is fixed in the blueprint. The conclusion of the rule outputs persistent facts `!MSK(msk_new, ~t_new)` and `!MPK(mpk_new, ~t_new)` (which bind those values to the corresponding epoch) and adds `mpk_new` to the adversary's knowledge base.

State "update sk" to `rule distribute_token` and `rule update_usk`

After an epoch transition, a user who already has a secret key needs an update value to update it for the new epoch. In the state diagram, the events "TTP creates key for user" and "user applies update value to old key" are subsumed in the state "update sk", because the diagram describes only the states for one user and one TTP. In the prover models, however, the effects of update value distribution to distinct users must be considered and

separating the two events allows for a cleaner model. Therefore, this state is divided in two rules, `rule distribute_token` and `rule update_usk`.

`rule distribute_token` This rule (line 61 in Appendix A) requires as input an epoch identifier t and the master secret key of t. The update value ("token") is computed in some way (which is individual to each approach and needs to be filled in for the placeholder). The rule outputs it as a persistent fact that binds it to an epoch and a user. Some approaches distribute (parts of) the update over an insecure channel, which is accounted for with another placeholder in the conclusion.

`rule update_usk` This rule (line 74 in Appendix A) requires as input an epoch identifier t_{new}, a persistent fact that records its connection to the previous epoch t_{old}, a user's key from epoch t_{old} and an update value for t_{new}. With an approach-specific formula, the user's key for t_{new} is computed and the result recorded as a persistent fact that binds it to t_{new} and the user ID.

State "use pk" to `rule encrypt`

All approaches are modeled as encryption schemes; therefore the rule is re-named to reflect that the public key is used to encrypt a message for a certain user.

In the premise, this rule (line 86 in Appendix A) takes as inputs mpk_t for the encryption epoch, a fresh message and an epoch identifier to specify the epoch for which the message shall be encrypted. The ID of the user for whom the message is encrypted must appear as a public value in the formula that specifies (for each approach individually) how a message is encrypted in the corresponding approach.

Choosing the message as a fresh value means that the same message can never be encrypted twice using this rule. Unless the adversary encrypts the same message again, effects from encrypting the same message to different users can not be discovered. The resulting cipher is published, which models sending it to the receiver over an insecure channel. There is no other output.

State "use sk" to `rule decrypt`

When modeling encryption schemes, the user's secret key is used to decrypt messages; the rule is re-named accordingly. This rule (line 99 in Appendix A) takes as input a cipher generated for an epoch t from the public channel, a user secret key and a master public key. The master public key is necessary in this rule to match the correct $usk_{ID,t}$ against the cipher, because the connection between them is established through msk_t. The code replacing the placeholders details how $usk_{ID,t}$ and cipher are matched against each other in a certain approach and computes the message that the cipher decrypts to. There is no output from this rule.

The restrictions in the model, which ensure the correctness of the epoch transitions as well as some of the TTP's and the users' execution policies, are explained in the next section.

6.3.2. Restrictions

The following restrictions model behavior of the TTP and the users that is not captured in the rules. They are found in lines 133 through 163 of the blueprint code.

Restrictions achieve two things:

1. Proofs become easier because the trace space is smaller.
2. The attacks excludes unrealistic scenarios. However, it is not necessary (and may even be detrimental) to exclude *all* unrealistic scenarios, as long as any attacks the prover finds do not rely on them.

Proving a lemma for more cases than necessary does not harm its validity. Some restrictions may be difficult for the prover to process and thus make proofs harder rather than easier to compute; therefore, skipping restrictions can be a sensible modeling option. Also, if the model is restricted too heavily, possible attacks may inadvertently be excluded.

The following set of restrictions strikes a balance between these goals and are explained below:

- SetupOnlyOnce
- End_of_Epoch_holds
- End_Epoch_Once
- USK_Gen_or_Upd_only_once_per_epoch
- GenUSK_only_if_not_member_in_previous_epoch
- CreateUpdate_and_GenUSK_not_in_the_same_epoch
- CreateUpdate_only_once_per_epoch
- Update_MSK_only_once_per_epoch
- Leak_USK_only_once_per_epoch

Applying the blueprint to a different approach may warrant a different solution.

SetupOnlyOnce

This restriction ensures that `rule setup` occurs only once. It is implemented as follows:

```
restriction SetupOnlyOnce:
"All #i #j. SetupOnlyOnce()@ i & SetupOnlyOnce()@ j ==> #i = #j"
```

This restriction effects that there is only one starting point. Otherwise, the system can have more than one *msk*. While this would allow modeling parallel sessions and effects between communication groups (given extra mechanisms to control group membership), it is out of scope for this work.

End_of_Epoch_holds

The rules given above are not enough to enforce an epoch transition where only one epoch exists at a time. When `rule advance_epoch` is executed, the epoch identifier for the ending epoch must not be used for key generation, updates or encryption anymore. Therefore, all rules that may only be executed in a specific epoch are labeled `Is_Epoch(~ t)` and `rule advance_epoch` is labeled `Epoch_ended(~t)`. The restriction states that for all (epoch identifiers) `t`, the label `Is_Epoch(t)` can only appear before the label `Epoch_ended(t)`:

```
restriction End_of_Epoch_holds:
"All t #i #j. Epoch_ended(t)@ i & Is_Epoch(t)@ j ==> #j < #i"
```

End_Epoch_Once

The existence of an epoch identifier is not modeled as a consumable fact but as persistent fact `!Epoch(~t)`. Therefore, the only condition to execute `rule advance_epoch` is always met as soon as the first epoch exists. The order of epochs must be unique, however: Time must be modeled as linear and no epoch can be the predecessor of two different epochs. The restriction `End_Epoch_Once` enforces that `Epoch_ended(~t)` can occur only once per epoch:

```
restriction End_Epoch_Once:
"All t #i #j. Epoch_ended(t)@ i & Epoch_ended(t)@ j ==> #i = #j"
```

USK_Gen_or_Upd_only_once_per_epoch

The formalization assumes that the TTP generates only one *usk* or update token per identity and epoch. The TTP's decisions on which keys and tokens are generated are abstracted in the formalization and in the model. Therefore, the model does not include an attack vector for fooling the TTP into generating more keys. The restriction enforces this policy through a label `Exists_Key`, which appears in `rule gen_usk` and `rule update_usk`

```
restriction USK_Gen_or_Upd_only_once_per_epoch:
"All I usk1 usk2 t #i #j.
Exists_Key(I, usk1, t)@ i & Exists_Key(I, usk2, t)@ j ==> #i = #j"
```

GenUSK_only_if_not_member_in_previous_epoch

Any user who was a member of the communication group in the previous epoch (i.e. legitimately held a valid key) must either receive an update or no key at all. New keys are distributed only to users for whom no key or update token was created in the previous epoch. The restriction enforces (using corresponding labels in `rule generate_user_key` and `rule advance_epoch`) that if a key is generated for a user in a certain epoch, they may not have received a key through `rule generate_user_key` or `rule update_usk` in the previous epoch.

```
restriction GenUSK_only_if_not_member_in_previous_epoch:
"All I t1 t2 #i #j.
Consecutive(t1, t2)@ #j & GenUSK(I)@ #i & Is_Epoch(t2) @ #i
==> not(Ex usk #k. Exists_Key(I, usk, t1)@ #k)"
```

CreateUpdate_and_GenUSK_not_in_the_same_epoch

Users who were legitimate members in the previous epoch only receive update tokens, not freshly generated keys, and those who (re-)join but were not members in the previous epoch only receive freshly generated keys but not updates. So updates and fresh keys are never created for the same user in the same epoch.

```
restriction CreateUpdate_and_GenUSK_not_in_the_same_epoch:
"All I t1 t2 #i #j. Consecutive(t1, t2)@ #j & Create_Update(I, t2)@
   #i
==> not (Ex #k. GenUSK(I)@ #k & Is_Epoch(t2)@ #k)"
```

This restriction excludes implausible attacks where an update token for `I` is leaked even though a fresh user key was generated for `I` in the corresponding epoch (such that an update token would not make sense for that epoch).

CreateUpdate_only_once_per_epoch

The TTP creates only one update for each user and epoch, and only distributes it once. Legitimate distribution errors (for example, technical issues with the private channel between user and TTP) are abstracted in this model; private channels are assumed to be perfect. The user is also restricted to updating their key only once per epoch and can only use update values distributed via a persistent fact. This reduces the attack surface since it makes update injections more difficult for the attacker. In the presented models, the results are not affected by this restriction, so it is safe to include it to simplify the scenarios in the counterexamples and for reduced runtime.

```
restriction CreateUpdate_only_once_per_epoch:
"All t I #i #j. Create_Update(I, t)@ #i & Create_Update(I, t)@ #j
   ==> #i = #j"
```

Update_MSK_only_once_per_epoch

The master key material is only updated by the TTP once per epoch. The restriction enforces this using label `Update_MSK(~ t)`, such that `rule update_msk` only be executed once per epoch.

```
restriction Update_MSK_only_once_per_epoch:
"All t msk1 msk2 #i #j. Update_MSK(msk1, t)@ #i & Update_MSK(msk2,
   t)@ #j ==> #i = #j"
```

Leak_USK_only_once_per_epoch

In the presented model, the adversary either does or does not know a key; it therefore makes no difference whether the same information is leaked repeatedly or taken from the adversary's knowledge base once it has been added to it. This restriction is added to reduce the trace space and make attack examples more straightforward.

```
restriction Leak_USK_only_once_per_epoch:
"All I t #i #j. LeakUSK(I, t)@ i & LeakUSK(I, t)@ j ==> #i = #j"
```

With these state rules and restrictions, the main building block of the revocation mechanism is established. The next section explains the lemmas that represent the security goals, which are formulated as statements about action facts, and Section 6.5 shows how the implementation of the revocation mechanism is refined to capture the adversary abilities and the mathematical dependencies.

6.4. The security properties

Security properties are formulated as "lemmas", i.e. first order logic statements about the traces. The basic structure most often follows the form:

"It holds for all traces that if a certain *label* appears at a "time point" `#i` on the trace, then the adversary has no knowledge of a secret value unless a certain attack occurs."

The statements can include multiple time points and labels both on the left and the right hand side of the implication. There are also implication-free statements about all (considered) traces or about the existence of traces with certain properties.

The security goals are the same for all models so that the different approaches can be compared as closely as possible. However, the codes differ slightly because the attacker model is not the same in all cases; for example, only the individual-token approaches contain a static *usk*, so the lemmas for the security goals contain some reasoning about static *usk* leakage that is absent in the other models.

The following sections explain the basic translation of the security properties into Tamarin code that is the same for all approaches. The section on Forward Security includes an example and a verbose explanation to showcase the code for such a lemma. The translations for the remaining security goals are explained more briefly and only reference the corresponding code in Appendix A.

Chapter 7 compares the exact conditions that need to be included in the lemmas for each approach to understand whether it achieves the property, thus addressing the placeholders.

6.4.1. Forward Security

According to Section 5.4, "Forward Security means that messages encrypted with a public key from an epoch t remain confidential even if a secret key of the same user from an epoch $t + x > t$ is compromised".

The following code example shows the corresponding Tamarin lines from the blueprint in Appendix A (explanation below):

```
230 lemma forward_security:
231         "All m I t1 t2 #i #j #k #o.
232         Sent(m,I)@ #i
233         & Is_Epoch(t1)@ #i
234         & Epoch_ended(t1)@ #j
235         & LeakUSK(I,t2)@ #k
236         & Is_Epoch(t2)@ #o
237         & #j < #o
238         ==> not(Ex #l. K(m)@ #l)
239         | (Ex #l. LeakUSK(I, t1)@ #l)
240         | (Ex #l. LeakMSK(t1)@ #l)
241         | (Ex #l. LeakUpdVal(I,t1)@ #l)
242         %other trivial leak"
```

For all traces and all variables `m`, `I`, `t1` and `t2` (which represent a message, an identity, and two epoch identifiers) and for all trace time points `#i`, `#j`, `#k` and `#o` the following holds:

Assume `m` is encrypted for `I` in epoch `t1` (so, the `Sent(m,I)` action fact and `IsEpoch(t1)` action fact happen at the same time point `#i`), epoch `t1` ends at some time point (which, per the restriction `End_of_Epoch_Holds`, happens after the message was sent). Assume further, that $usk_{I,t2}$ is leaked for an epoch `t2`, which occurs after the end of epoch `t1` (`#j` $<$ `#o`). Then there exists no time point `#l` at which the adversary knows `m` (denoted by `K(m)`).

To get functional code, the placeholders (lines prefaced with `%` and highlighted in yellow) must be replaced by code to exclude attacks in which the leak of $usk_{I,t2}$ plays no role (and which are therefore not valid as counterexamples for Forward Security).

6.4.2. Post-Compromise Security

Section 5.4 states that "Post-Compromise Security is achieved if compromising $usk_{ID,t-x}$ from an epoch $t - x < t$ gives an attacker no advantage when trying to decrypt a message that was encrypted for an identity ID and an epoch t".

The corresponding `lemma post_compromise_security` (line 246 in Appendix A) therefore states that for all traces, (messages) `m`, (user identities) `I`, (epochs) `t1` and `t2` and all trace time points `#i`, `#j` and `#k` the following holds: Assume usk_{I,t_1} is leaked, epoch `t1` ends and afterwards, a message `m` is sent to `I` in an epoch `t2`. Then there exists no time point `#l` at which the adversary knows `m` as a result of this leak (and other attacks are excluded).

6.4.3. Decryption Key Exposure Resistance Forward

Section 5.4 states that "A mechanism achieves Decryption Key Exposure Resistance Forward if the adversary can not learn $usk_{ID,t}$ without compromising the key itself or msk_t, even if $usk_{ID,t+x}$ for any later epoch $t + x$ is compromised."

The corresponding lemma (line 266 in Appendix A) therefore states that for all traces, (user identities) `I`, (epochs) `t1` and `t2`, all `usk1` and `usk2`, and all trace time points `#i`, `#j` and `#k` the following holds: Assume there exists a key `usk1` that is bound to `t1` and identity `I` (i.e. it is the result of a user key generation or a user key update in epoch `t1`, the two rules in which the corresponding label appears). Assume further that `usk2` exists for `I` at a later time point `t2` (which, per the restrictions, implies that `usk1`≠`usk2` and `t1`≠`t2`), and `usk2` is leaked at some time point. Then there exists no time point `#l` at which the adversary knows `usk1` as a result of this leak (and other attacks are excluded).

6.4.4. Decryption Key Exposure Resistance Backward

Section 5.4 states that "A mechanism achieves Decryption Key Exposure Resistance Backward if the adversary can not learn $usk_{ID,t}$ of an epoch t without compromising the key itself or msk_t, even if $usk_{ID,t-x}$ from a previous epoch $t - x$ is compromised."

The corresponding lemma (line 286 in Appendix A) therefore states that for all traces, (user identities) `I`, (epochs) `t1` and `t2`, all `usk1` and `usk2`, and all trace time points `#i`, `#j` and `#k` it holds that: Assume there exists a key `usk1` that is bound to epoch `t1` and identity `I` (i.e. is the result of a user key generation or a user key update in epoch `t1`). Assume further that `usk2` exists for `I` at a later time point `t2` (which, per the restrictions, implies that `usk1`≠`usk2` and `t1`≠`t2`), and `usk1` is leaked at some time point. Then there exists no time point `#l` at which the adversary knows `usk2` as a result of this leak (and other attacks are excluded).

6.4.5. Collusion Resistance

Section 5.4 states that "[a]n update mechanism is called collusion resistant if it does not allow any number of users to decrypt a message together that none of them should be able to decrypt on their own."

The lemma (line 306 in Appendix A) therefore requires that for all traces and all variables `m`, `I` and `t` (which represent a message, an identity and an epoch identifier) and for any trace time point `#i` the following holds: Assume message `m` is encrypted for `I` and epoch `t`, but no fresh key or update token was generated for identity `I` in epoch `t`. Then the adversary can

```
rule leak_msk:
[ !MSK(msk,~t) ]
--[ LeakMSK() ]->
[ Out(msk) ]
```

Listing 6.3: Code that allows leaking *msk* to the adversary's knowledge base

not know `m` unless the master secret is leaked. Only attack scenarios that include information from a third user (who is neither the sender nor the recipient of the encrypted message and not the TTP) are (in this work) considered proof that Collusion Resistance does not hold or only conditionally holds in the modeled approach. The adversary is then understood to collude with this third user.

6.5. The adversary model

The following paragraphs describe how each of the adversary's attack vectors formalized in Section 5.5 is modeled.

6.5.1. Adversary knowledge

As explained in Section 2.7, an adversary can intercept, delay, modify, replay, delete and create messages sent over an unprotected channel. In TAMARIN, values appear on the untrusted channel either as fresh values, which the adversary can create any time and arbitrarily often, or as rule outputs. The adversary can inject any messages in the protocol via an `In(...)`-fact. She can deduce facts using all functions and equations. She can use all rules with values she knows. The rules for derivation and injection of adversary knowledge appear in the manual; for this work it suffices to know the action fact `K(x)`, which denotes the adversary knowing `x`.

6.5.2. Corrupting Secret Values

If values are only passed as arguments of persistent facts, the adversary can not know them, so they are secret. To model the corruption of secret values, the code needs to contain rules that specifically allow the adversary to learn them. Listing 6.3 shows the rule with which msk_t is leaked to the adversary's knowledge base: If a persistent fact for the master secret in epoch t exists, then the master secret is sent to the public channel via `Out()`.

Analogous rules exist to leak $usk_{ID,t}$ or the update value $upd\_val_{ID,t}$ for a user ID in epoch t. A placeholder in the blueprint indicates that models may need to include more leakage lemmas if other secret values exist. For example, the individual-update approaches contain a secret, static user key $susk_{ID}$ which is used to update $usk_{ID,t}$; a leakage rule for this value is therefore added to these models.

The corresponding label in every rule allows the model to control for various attack scenarios in the lemmas, so the conditions of achieving a security goal are transparent and comparable.

```
usk = h($I, ~msk, ~t)
```
(a) key renewal

```
susk = h($I, ~msk)
token = token($I, ~msk,~t)
usk = < susk, token >
```
(b) individual token, separate

```
susk = h($I, ~msk)
token = token($I, ~msk,~t)
usk = h(susk, token)
```
(c) individual token, re-randomized

```
usk = usk_der($I, msk_curr)
```
(d) universal token

Figure 6.3.: Code that replaces the placeholder for the generation of user secret keys in each of the four models

6.5.3. Dependency between msk_t and MPK_t

The rules `setup` and `update_msk` specify that and how MPK_t is computed from msk_t. In all considered approaches, deriving msk_t from MPK_t is supposed to be hard (because despite MPK_t being public, msk_t should remain secret). The computation must therefore be modeled through a function which is not invertible for the adversary. Also, the encryption/decryption equation must uniquely match the msk_t used for the $usk_{ID,t}$-derivation with the MPK_t used for the encryption, so the derivation must happen through a function with which the match can be enforced. Therefore, the dependency between msk_t and MPK_t is modeled via the built-in function `pk()` (which is irreversible and suited for pattern matching) in all models: `mpk = pk(msk)`. Only in the universal token approach, the msk is not a fresh value but modeled through a function `msk_der`: `msk = msk_der(~msk_fresh)`. This is necessary to model msk_t-updates in such a way that extracting old values from a known msk_t is impossible. Going forward, the argument `x` in `msk_der(x)` will be called "msk-exponent"[2] for readability.

6.5.4. Dependency between ID, msk_t, t and $usk_{ID,t}$

In `rule generate_usk`, a placeholder for the user key appears:

```
27 %formula for the computation of usk
```

The following paragraphs explain the dependency between ID, msk_t, t and $usk_{ID,t}$ as reflected in `rule generate_usk`. The dependency is also partly reflected in the update mechanism, so further details are found there (Section 6.5.6).

As explained in Section 3.2, exactly modeling the mathematics of identity-based encryption- and signature schemes in Tamarin is difficult. Therefore, the exact functions are abstracted. The abstractions must, however, capture the essential mathematical properties of the underlying functions as faithfully as possible so attack vectors can be recognized. Figure 6.3 shows the corresponding Tamarin-code to replace the placeholder in each approach.

[2] This naming alludes to the schemes analyzed in [Gug20], where msk_t is the exponent of MPK_t and updated via exponent multiplication.

The following properties are captured:

key renewal It must be ...

- ... easy to compute $usk_{ID,t}$ given ID, t and msk_t, but hard to compute given only ID and t, and
- ... hard to compute msk_t from $usk_{ID,t}$.

Using a one-way function (namely the built-in hashing `h(·)`) enforces both. It also models the $usk_{ID,t}$ as non-malleable, meaning a change in `msk` can not be applied to `usk` without re-computing the complete hash. Note that important properties of a real function for computing $usk_{ID,t}$ may be missing here; hashing is not necessarily a secure way to compute a secret key.

individual token (separate) The formulas ensure that ...

- ... the static key is computed independent of the epoch, and easily computed given `ID` and `msk`, but extracting `msk` from it is impossible (through the properties of the hash function).
- ... the token depends on msk_t, the user's identity and the epoch. As long as no further properties of the function `token` are modeled, the computation is a one-way function. That means, it is easy to compute `token` given ID, t and msk_t, but hard to compute given only ID and t, and extracting `msk` is impossible.
- ... given `susk` and a `token` for an epoch, it is easy to compute `usk`. Given `usk`, both `susk` and `token` can be easily extracted (the tuple function `<·,·>` allows this in TAMARIN).

individual token (rerandomized) The formulas to compute `susk` and `token` are the same as for the **individual token (separate)**-approach, so the properties are the same, too. The function for computing `usk` is different, though: Instead of a function where `susk` and `token` can be extracted given `usk`, the combination of the two values is re-randomized with a one-way function (again represented by a hash-function) to yield `usk`. This way, knowing `usk` implies no knowledge of `susk` or `token`.

universal token The connection between epoch and $usk_{ID,t}$ is not explicit in the universal token approach. Instead, the connection is made by ensuring both msk_t and $usk_{ID,t}$ share a certain value that is freshly chosen for each epoch. Therefore, the function `usk_der`, with which `usk` is derived, does not take an epoch identifier as an input, but only the user's identity and the current msk-exponent. The function also has important properties for enabling updates with a universal token; see Section 6.5.6 for details.

Since the msk-exponent is hidden in `msk_der` (even in leaks, because the adversary can otherwise extract too much information from the tuple that represents the exponent), the adversary must be given the ability to compute $usk_{ID,t}$ from msk_t directly (see `rule msk_yield_usk` in Appendix E, line 128).

6.5.5. Dependency between msk_{t_1} and msk_{t_2}

In `rule update_msk` appears a placeholder for updating msk_t (computing `msk_new`).

47 `msk_new = %formula for msk update`

From `msk_new`, the updated MPK_t is computed as `pk(msk_new)`.

For the approaches using key renewal and individual updates, the master key material is never updated. Therefore, `rule update_msk` is left out of these models to reduce complexity.

In the universal token approach, $\sim$ `upd_val` is applied to the previous master key by appending it to the msk-exponent: `msk_new = msk_der(<~ upd_val, msk_old_inside>)` The `<·,·>`-function represents the multiplication of the old exponent with the update value. This built-in function allows nesting updates: A tuple `<update, old_value>` can be extended to the left with new update values arbitrarily often while keeping the same tuple structure (which is necessary for pattern matching in other rules). This reflects that each multiplication yields a value that can itself be multiplied with other values, which is an important property that is otherwise tedious to implement in Tamarin.

Note that without any further rules or equations that describe mathematical properties of `msk_der(·)`, there are no more attack vectors than leaking the values. To limit the model's complexity, division by a previous msk_{t-x} or division/multiplication by update values is not modeled for `msk_der(·)` although it is possible in [Gug20]; see Section 7.3.8 for a discussion.

6.5.6. Dependency between usk_{ID,t_1} and usk_{ID,t_2}

For consecutive epochs t_1 and t_2, `rule distribute_token` and `rule update_usk` establish the mathematical connection between a user's respective keys. There are three placeholders:

63 `update_value = %formula for the computation of the update value` in `rule distribute_token`

70 `%other output for token distribution` in `rule distribute_token`

76 `usk_new = %formula for the computation of the usk update` in `rule update_usk`

In the four approaches, these are filled as follows:

key renewal Key updates happen by key renewal, so for a "token", the TTP generates an entirely new key, treating the epoch identifier as a part of the user's identity. Computing a user key from another user's key must be hard; so in this update mechanism, it must be hard to compute the user key for a consecutive epoch from a previous one. This property is modeled by the hash function with which keys are generated. Therefore, the placeholder replacements are straightforward (line numbering given for the corresponding code in Appendix B):

- 61 `update_value = h($I, msk, t)`, so the update value is a fresh key for the given identity, msk and epoch
- 67 *(no further output in `rule distribute_token`)*
- 72 `usk_new = upd_val`, so the new user key is instantiated with the fresh key that is passed to the rule as an update value

individual token (separate and rerandomized) For the individual approach, a fresh individual token is generated for each identity and epoch (`rule distribute_token`). It is also an important claim of the individual approach that update tokens can be published without harming security; therefore, the token is not only output as a persistent fact but also to the insecure channel over which it is sent to the users. The placeholders are filled with (line numbers refer to Appendix C):

51 `update_value = token($I, ~msk, ~t_new)`

58 `Out(update_value)`

Given $susk_{ID}$, the update is applied with the formula `usk_new = <susk, upd_val>` in the *separate* approach (Appendix C, line 65) and with `usk_new = h(susk, upd_val)` in the re-randomized one (Appendix D, line 65). Like for the key generation, this reflects that in the separate approach, the token is seamlessly swapped, and in the rerandomized approach, the key needs to be re-computed for re-randomization.

universal token For the universal token approach, two properties need to be reflected in the code:

- Given a token with which `msk` is updated for a new epoch, it must be possible to update `usk` knowing only the token and not `msk`.
- As described in [Gug20], the proposed mechanisms allow reversing the update given $usk_{ID,t}$ and an update token for epoch t, so that usk_{t-1} can be computed. Although the described mechanisms are only for signatures (and no corresponding encryption mechanisms are known), it is plausible that this property would translate to encryption, so the model must capture it as well. (Likely, the implications for security translate back to signatures as well.)

The model needs to reflect them in the two rules used for updating user keys, but also with additional rules that allow the adversary to work with leaked values in the same way.

The placeholders for the token- and $usk_{ID,t}$-computation are filled as follows (line numbers refer to Appendix E):

65 in `rule distribute_token`:
```
msk_new = msk_der(<~upd_val, msk_old>)
update_value = ~upd_val
```
That means that the TTP selects the same token for distribution that is used before to derive the new msk_t.

77 in `rule update_usk`:
```
usk_old = usk_der($I, old_value)
usk_new = usk_der($I,<~upd_val, old_value>)
```

There is no additional output for `rule distribute_token` since the token is sent over a private channel.

The two additional rules are:

`rule usk_der_math` , which allows updating a given old user key with a given update value (line 116):

```
rule usk_der_math:
        [ In(usk_der($I,<~exponent1,exponent2>))
        , In(~upd_val)]
        --[USK_der()]->
        [ Out(usk_der($I,<~upd_val, ~exponent1, exponent2>))]
```

rule usk_diff_math , which allows reversing a known update in an updated key (line 122):

```
rule usk_diff_math:
        [In(usk_der($I,<~exponent1,exponent2>))
        ,In(~exponent1)]
        --[ USK_diff() ]->
        [Out(usk_der($I,exponent2))]
```

6.5.7. Dependency between Keys, Plaintext and Ciphertext

The model must specify how the moving parts of the encryption interconnect with the parameters necessary for decryption to reconstruct the underlying message from a cipher text. The specification must allow the adversary to perform the corresponding steps if it knows all necessary values. The corresponding placeholders in the blueprint (code lines given) are:

6 `%the encryption/decryption equation of each model [...]` in `equations`

88 `%formula for the encryption of message m` in `rule encrypt`

101 `%pattern match for the usk` in `rule decrypt`

102 `%pattern match for the cipher` (ibid.)

103 `%computation formula for decrypted message` (ibid.)

In the model for key renewal, the equation that defines the functions `ibenc` and `ibdec` (identity-based `encryption` and `decryption`) is the following: `ibdec(ibenc(m, I, pk(msk), t), h(I, msk, t)) = m`
It is almost identical for the two models for the individual approach, with only `h(I, msk, t)` substituted by the respective formulas for computing $usk_{ID,t}$. For the universal approach, not only is `h(I, msk, t)` replaced with `usk_der(I, msk)`, but `t` does not appear in the equation at all; `msk` is time-dependent in this case, so an explicit reference to the epoch identifier is unnecessary.

The placeholders for the encryption and decryption mirror the equation, using those parts of it that are necessary to produce the intended result in the lemmas.

6.5.8. Dependency between Epoch Identifiers

Many approaches propose using time stamps to derive keys and decrypt messages. Time stamps have an underlying mathematical logic; however, this logic is never conceivably used in any of the considered approaches and the values are mere placeholders with no inherent connection. At the same time, introducing a mathematical connection (like summation) to the epoch identifiers is not straightforward in Tamarin. Therefore, this dependency is not modeled here.

6.6. Sanity checks

The model must include sanity checks to ensure that the models makes sense and faithfully represent the mechanisms in question, and that the exceptions expressed in the security lemmas are minimal.

6.6.1. Meaningful representation

It needs to be shown that messages can be encrypted and decrypted, both before and after an update (to check the functionality), and that more than one user can have keys in the same epoch (to ensure that effects between users can occur). As a special sanity check for this work, there is a lemma to prove that the model faithfully captures the formalization given in Chapter 5 by constructing a trace that mirrors the minimal example (Figure 5.1).

It is also necessary to check that the left-hand sides of lemmas that contain implications are not empty; otherwise, they are trivially true. However, this happens as a side-effect of identifying the minimal assumptions under which the lemmas hold, which is discussed in Section 7.2. Therefore, those checks are omitted in this section.

lemma can_receive

This lemma states that there exists at least one trace in which the label `Receive(~ m)` appears (indicating that some user received a message through `rule decrypt`), even if no $msk_{t'}$, $usk_{ID',t'}$ or $token_{ID',t'}$ is ever leaked for any epoch t' or identity ID'.

lemma can_receive_after_update

This lemma ensures that an updated key can be used for decryption.

lemma two_users_can_have_keys_in_same_epoch

Some effects may only be observed if several users are present in the system. Therefore, this sanity check ensures that at least two users can simultaneously have keys in the modeled system.

lemma minimal_example

To ensure that every model faithfully implements the minimal example illustrated in Chapter 5, every model includes a sanity check lemma that proves the existence of a corresponding trace. The lemma allows two interpretations of the user who receives a freshly generated key in the fourth epoch: They may be instantiated with user A or with a new user D. Highly specific lemmas like these are uncommon because it is usually sufficient to include general sanity checks. Successful proof of the `minimal_example` lemma implies the veracity of the other sanity check lemmas; they are still included in the code to demonstrate what more common sanity check lemmas for such a model are.

6.6.2. Minimal assumption checks

All lemmas include exceptions for cases in which trivial attacks occur, i.e. the adversary learns a value but the attack is not a counterexample for the security lemma because it does not exploit the attack vector on which the lemma focuses. Section 7.2.2 and Section 7.2.3 detail the overall and approach-specific trivial attacks that need to be excluded to prove the lemmas or find non-trivial attacks. Note that not all possible trivial attacks need necessarily be excluded if at least one non-trivial attacks are found, because this suffices to conclude that a security notion is not achieved in that case.

6.7. Modeling summary

The blueprint model and the approach-specific adjustments that are described in this chapter cover all parts of the formalization: The key management and revocation process (state rules and restrictions), the adversary knowledge (equations and symbolic dependencies in rules plus leakage rules), and the security goals (lemmas). The sanity checks help ensure that formalization and implementation are congruent. Thus, the chapter answers **RQ 3)** ("How can the formalization be translated for an automated reasoning tool?")

Chapter 7 discusses running the concrete models and the results that they yield.

7. Results and Evaluation

This chapter reports and discusses the results from running each of the four code variants presented in Chapter 6 in Tamarin.

Section 7.1 describes the setup and runtime of each model execution. It also explains that (and where) changes or additions to the code compared to the blueprint are necessary so that Tamarin runs in reasonable time and with the given memory. The code that yields the results described in this chapter is available online.

Section 7.2 describes and compares the results for each approach and the (minimal) conditions under which the security goals are met.

Section 7.3 discusses how the results compare to findings from the literature, to what extent the results reconcile the notions "Forward/Post-Compromise Security" and "Decryption Key Exposure Resistance", and the value of the notion "Collusion Resistance". Limits of the findings are discussed as well.

7.1. Setup and Model execution

The results were obtained by running the codes on a virtual machine with 40 vCPUs (model: Intel(R) Xeon(R) Silver 4316 CPU @ 2.30GHz) and 180 GB of main memory. The virtual machine ran at "LRZ compute cloud"[1], a service hosted by Leibniz Rechenzentrum (LRZ, Leibniz Supercomputing Centre) in Garching. The models were executed running Tamarin version 1.6.1[2].

For some lemmas in some models as described in Chapter 6, the execution takes more than eight hours or does not terminate at all. To enforce termination and shorten runtimes, the models are adjusted by

- adding helper lemmas for the key renewal- and universal update models to break unresolved loops
- removing the (inconsequential) time dependency of the master keys in the individual update models
- limiting the number of epochs and the number of user key up- and downgrades the attacker can compute in the universal update model

The following paragraphs describe the adjustments in more detail:

key renewal The epoch transitions and the related user- and master key updates create loops, making it difficult for Tamarin to infer `rule setup` as the starting point. While the sanity-check lemmas (except for `lemma minimal_example`) terminate within a few minutes, `lemma forward_security` does not terminate after several hours in

[1] `https://doku.lrz.de/compute-cloud-10333232.html`
[2] (released Aug 23, 2021 on `https://github.com/tamarin-prover/tamarin-prover/releases`)

the key renewal model. The following lemma remedies this ([sources] indicates that Tamarin uses the result to deconstruct sources for other lemmas):

```
lemma msk_mpk_never_change [sources]:
"All msk mpk #i.
Is_MPK_MSK(msk, mpk)@ #i ==> (Ex #j. Origin_MPK_MSK(msk,
    mpk)@ #j)"
```

Tamarin proves this lemma in under a minute and re-uses the statement; namely, that for any master key pair that exists at any given time point, there must exist an execution of `rule setup` (the only rule where the label `Origin_MPK_MSK()` occurs) featuring the same key pair. As described before (Section 6.5.5), the key update function is eliminated from the model without a loss in functionality, making the master key material epoch-independent. The corresponding code terminates proving the same lemmas.

individual update (both) For the individual updates, the loop from the master key updates introduce so much complexity that running `lemma minimal_example` does not terminate. When the time-dependence of the master key material is left out of the model (so, `rule update_msk` is removed, such that *msk* and *MPK* are only created in `rule setup` and never updated), all lemmas can be verified or falsified within 35 minutes. Since `rule update_msk` only hands over the same master keys to the next epoch, this adjustment is still a faithful representation of the underlying approaches.

universal update For the universal approach, the complexity from the loops is more difficult to remedy, since master key material updates have an essential role in the approach.

Two adjustments are necessary to run the code in reasonable time or have the execution terminate at all:

- The model is restricted to four epochs: This reduces the search space for the traces drastically and loops that Tamarin can not resolve otherwise are simply limited in their length. Four epochs are enough to add a member, update their key, revoke their key, skip one epoch, and then add them again as a member. It is therefore plausible that within four epochs, Tamarin will detect any procedural attacks in the model. However, the restriction takes away the generality of the lemmas and where no attacks are detected, conclusions must be drawn more carefully.

  A sources-lemma supports this restriction: When there is a master key update, it must be either the first, second or third update.

- The derivation of new values from known user secrets via `rule usk_der` and `rule usk_diff` must also be limited to reduce the amount of cases that Tamarin can derive from them; otherwise, an infinite loop is possible. Therefore, either rule may only be executed four times. The number is chosen heuristically to strike a balance between allowing the model to combine several computations across epochs and users and reducing the complexity to have the models run in reasonable time.

Even with these changes to the **universal update** model, Tamarin can not automatically deduce the source of some facts (which results in so-called *partial deconstructions*). This makes it much harder, often impossible, for Tamarin to prove lemmas. If the program terminates, the resulting proofs and counterexamples are correct.

Table 7.1.: Approximate time that it takes the run of a lemma or set of lemmas (row) in a model (column) to terminate. Run times of less than ten seconds (< 10 s, very light green), between ten seconds and one minute (< 1 min, very light green), or between one and five minutes (< 5 min, light green) are vigorously rounded with respect to these three categories; higher values are rounded up to the next minute and printed in deep green. The values refer to run times on a virtual machine with 40 vCPUs and 18 GB.

lemma	key renewal	indiv. token, separate	indiv. token, re-randomized	universal token
sources	< 10 s	—	—	< 10 s
sanity checks	< 10 s	< 10 s	< 10 s	< 10 s
minimal example	**12 min**	**21 min**	**24 min**	**1 h 31 min**
Forward Security	< 10 s	< 1 min	< 10 s	< 1 min
Post-C. Security	< 10 s	< 10 s	< 10 s	< 1 min
DKER Forward	< 10 s	**< 5 min**	< 10 s	**7 min**
DKER Backward	< 10 s	< 10 s	< 10 s	< 1 min
Collusion Res.	< 10 s	< 10 s	< 10 s	< 10 s
Minimal assumption checks for:				
Forward Security	< 1 min	**< 5 min**	< 1 min	< 1 min
Post-C. Security	< 1 min	< 1 min	< 1 min	< 1 min
DKER Forward	**< 5 min**	**< 5 min**	**< 5 min**	**< 5 min**
DKER Backward	< 1 min	< 1 min	< 1 min	< 1 min
Collusion Res.	< 10 s	< 10 s	< 10 s	< 1 min
total runtime	**14 min**	**31 min**	**26 min**	**1 h 44 min**

Table 7.1 shows approximate run times for each adjusted model. The rows are divided in three parts: The first shows the runtimes for sources-lemmas, sanity checks, and the *minimal example* described in Section 5.2. The second shows the runtimes for checking the security goals, and the third, the minimal assumption checks for the security goals. The measurements refer to runs using the default heuristic and the **autoprove** option with default proof-depth (method "a." in the graphic user interface).

The table shows that in all models, `lemma minimal_example` takes longest to run, with run times between 12 minutes in the `renewal`-model up to roughly one and a half hours in the `universal token`-model. The remaining lemmas, including trivial attack checks, take up to 13 minutes in total to prove or disprove. On average, lemmas where counterexamples were found took longer to complete (namely the minimal assumption checks and the lemmas for the individual token update, separate, and the universal token update).

Note: Readers who intend to experiment on the code are advised that during the experimental phase of this work, variations of these models repeatedly ran for up to four hours to yield results. Many variations of the `universal token` model in particular are difficult for the prover to reason about and single lemmas can take hours to terminate. Models were never run for more than 48 hours, and usually not for more than five hours before interrupting the process (under the assumption that the model could not terminate for this lemma).

Leaving out certain attacker capabilities to check a lemma in a simpler model can be fruitful for finding attacks; e.g., leaving out `rule usk_diff_math` can yield a counterexample in which the function is not needed, but where including the function extends the reasoning options too much for the analysis to terminate.

7.2. Results of running the models in Tamarin

This section explains proofs and attacks found for the security lemmas in each model. For readability, the names of four lemmas are abbreviated in the following section:

fs for `lemma forward_security`

pcs for `lemma post_compromise_security`

dker_f for `lemma decryption_key_exposure_resistance_forward`

dker_b for `lemma decryption_key_exposure_resistance_backward`

Table 7.2 shows a) that both the key renewal approach and the individual token approach with re-randomization achieve (✓) all considered goals, b) that the individual token approach with separated keys fails (✗) to achieve any goal except Collusion Resistance, and c) that the universal token approach has none of the desired properties. This conclusion and the underlying attacks against the two vulnerable approaches are explained in more detail in the following sections, first discussing trivial attacks that are disregarded for the assessment and then discussing the non-trivial attacks that support it.

7.2.1. Sanity checks for meaningful representation

In all models, the sanity checks succeed. For example, checking `lemma can_receive` for the key renewal model yields the trace displayed in Figure 7.1.

The boxes in various shades of green all represent rule executions. The white bubbles represent adversary actions.

The trace illustrates the following process:
The top-most box shows `rule setup`, which takes fresh values ~`msk` and ~`t` as inputs and produces

- the persistent fact `!Epoch(~t)` (which is passed to the green box directly below, as indicated by a gray arrow)
- the persistent fact `!MSK(~msk)` (also passed to the green box directly below, again indicated by a gray arrow)
- the `Out(pk(~t))`-fact, which is added to the adversary's knowledge base (as indicated by a red arrow pointing to the left-most white bubble in the top row of bubbles)
- the `Out(pk(~msk))`-fact, which is added to the adversary's knowledge base (again indicated by a red arrow, this one pointing to the white bubble in the center of the top row of bubbles)
- the persistent fact `!MPK(~mpk)` (which is passed to the olive-green box on the bottom, as indicated by a gray arrow)

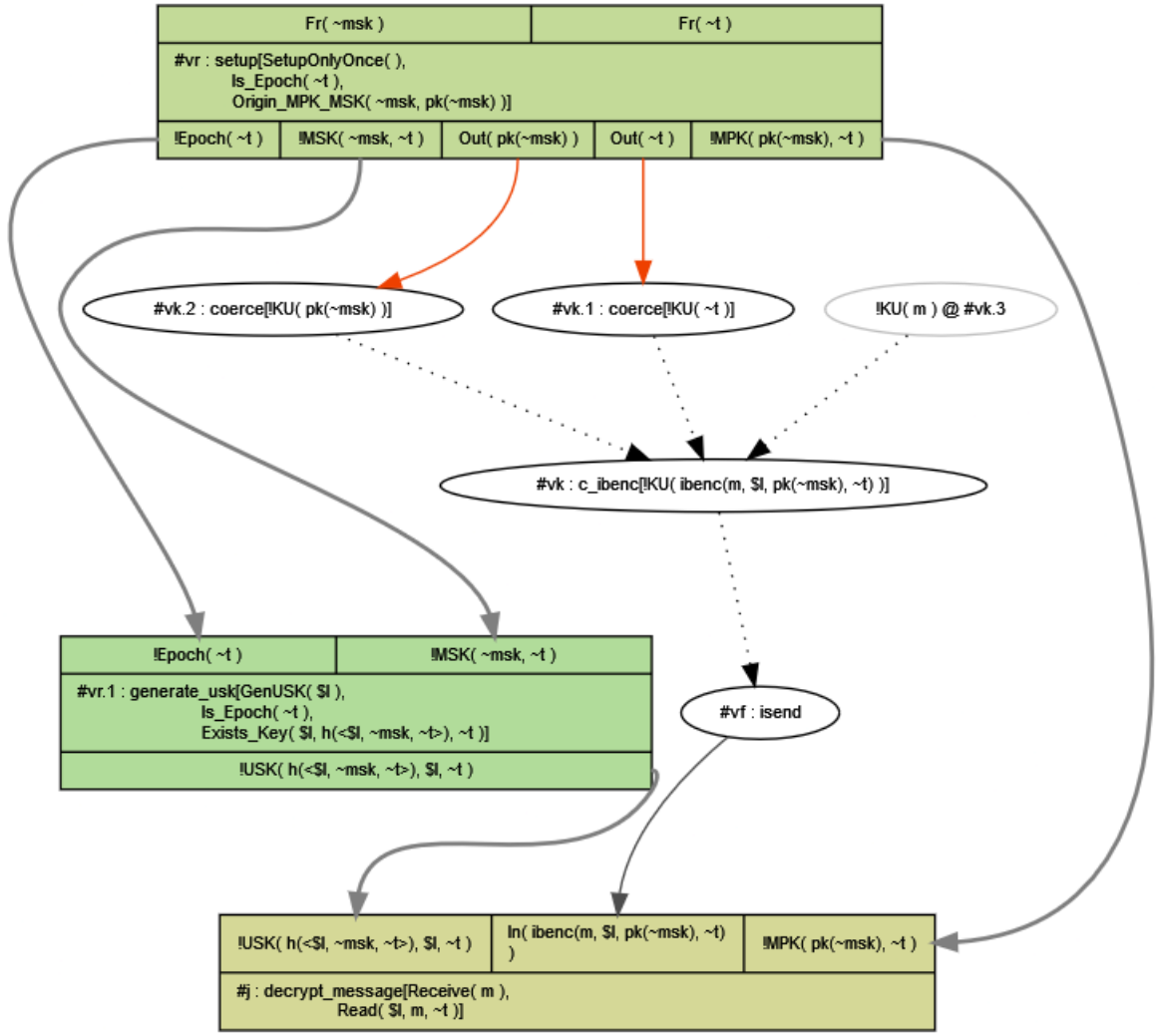

Figure 7.1.: Illustration of a trace that TAMARIN finds to prove `lemma can_receive` in the key renewal-model (graphic adapted from TAMARIN-output to fit the page). The boxes in shades of green represent rule executions (with inputs in the top row, rule names and labels in the middle row, outputs in the bottom row). White bubbles represent adversary knowledge and -actions. The arrows represent fact transfer (gray: from rule to rule; red: to the public channel; black/dotted: by the adversary). Details in the text.

Table 7.2.: Each entry indicates whether running a model *(column)* in TAMARIN shows that it has a property *(row)* (✓) or not (✗).
Superscript symbols indicate the corresponding attack vector described below:
I: A *usk* leak for any epoch makes other epochs insecure because the static key, with which all keys are derived, can be derived from it. Since the update tokens are published for each epoch, once an attacker has a static key, they can construct keys for any epoch for which an update token is published. Thus, no security goal that presumes the leak of a user key (= decryption key) is achieved.
V: A *usk* leak for an epoch t can be exploited to compromise the user secret for a previous (following) epoch $t-1$ ($t+1$) provided that an attacker also compromises the universal update token for t ($t+1$); in this case, the mathematical characteristics allow downgrading (updating) the user's key material, and messages sent before (after) the leak are not secure.
X: A user (re-)joining the group in an epoch t (or an attacker leaking their key) can read messages encrypted for them for the previous epoch (in which they were not a member) by colluding with another user and applying the other user's update to the leaked key.

lemma	key renewal	individual token (separate)	individual token (re-randomized)	universal token
Forward Security	✓	✗$^{\mathbf{I}}$	✓	✗$^{\mathbf{V}}$
Post-comp. Sec.	✓	✗$^{\mathbf{I}}$	✓	✗$^{\mathbf{V}}$
DKER Forward	✓	✗$^{\mathbf{I}}$	✓	✗$^{\mathbf{V}}$
DKER Backward	✓	✗$^{\mathbf{I}}$	✓	✗$^{\mathbf{V}}$
Collusion Res.	✓	✓	✓	✗$^{\mathbf{X}}$

Below `rule setup`, two arrows, each pointing to a white bubble, illustrate that the adversary receives the values published by the TTP in the setup of the session. The white bubble with a gray outline on the top right, labeled "`!KU(m) @ #vk.3`", illustrates that the adversary generates a fresh value `m` at a time point "vk.3". The facts `!KU(...)` represent adversary knowledge. Prefix "#" indicates time points (as in the rules.) The dotted arrows pointing to the central white bubble below the first three, illustrate that the adversary encrypts `m` using *MPK* and the epoch identifier `~t` with the (public) function `ibdec` and sends them over an insecure channel to a user (fourth dotted arrow and "isend" bubble). The green box in the middle represents `lemma generate_user_key`, which receives inputs from `rule setup` as described.

It outputs a persistent fact `!USK(h($I, ~msk, ~t),$I, ~t)` (indicating that $usk_{ID,t}$ was created for a user ID), which is passed to the olive-green box on bottom (gray arrow).

To the lower far left, the green box represents `lemma decrypt`, which receives the encrypted message from the insecure channel and two persistent facts from `rule setup` and from `rule generate_user_key`. It has no outputs, but only generates the events `Receive(m)` and `Read($I, ~m, ~t)`.

This process proves the existence of a trace in which the label `Read(~m)` appears and in which no secret values are leaked.

Minimal-assumption checks for the security lemmas validate the exceptions made for trivial attacks; Section 7.2.2 explains them in detail.

Table 7.3.: Overview of weaknesses found for each model. These weaknesses allow trivial attacks against the secrets of a user `I` for an epoch `t` with respect to both `dker_f` and `fs` (^{f}), or `dker_b` (^{b}), or all security properties (no superscript).
`R` is another user, `x` indicates additional epochs.

key renewal	individual token (separate)	individual token (re-randomized)	universal token
`LeakMSK(x)` $\forall x$ `LeakUpdVal(I,t)`	`LeakMSK(x)` $\forall x$ `LeakSUSK(I)` ^{f}`LeakUSK(I,t-x)` $\forall x > 0$	`LeakMSK(x)` $\forall x$ `LeakSUSK(I)`	^{b}(`LeakUpdVal(R,t)` $\wedge$ ... ... `LeakMSK(x)`) $\forall x > 0$

7.2.2. Overall trivial attacks

All considered security goals require a user I's key $usk_{ID,t}$ for epoch t to remain secret or require a message m encrypted for user I and epoch t to remain confidential. Two attacks are considered trivial for all mechanisms, as they are always possible:

`LeakUSK(I,t)` A direct compromise of the user key in question, which trivially allows the attacker to learn the key and to decrypt corresponding messages.

`LeakMSK(t)` A direct compromise of the master secret key with which $usk_{ID,t}$ can be directly derived and allows an attacker to learn the key and decrypt messages.

7.2.3. Approach-specific trivial attacks

Some trivial attacks found by TAMARIN exploit approach-specific attack vectors, as summarized by Table 7.3 in the column for each approach and explained in more detail below:

Key renewal There are two special attack vectors for this approach:

> **`LeakMSK(x)` for any epoch `x`** Since the master secret never changes in this approach, leaking it for one epoch means leaking it for every epoch. In the considered model, the resulting attacks are still trivial because for `fs`, `pcs`, `dker_f` and `dker_b`, they are independent of the assumed leak, and for Collusion Resistance, the master secrets are assumed to be secure for all epochs anyway.
>
> `LeakUpdVal(I,t)` In this approach, leaking the update value directly reveals the user key. So the attacks on the model are trivial with respect to `fs`, `pcs`, `dker_f` and `dker_b`.

Individual token (both) There are two special attack vectors for both variations of the individual token approach:

> **`LeakMSK(x)` for any epoch `x`** With the same reasoning as for the key renewal approach, leaking any master secret keys is considered a trivial attack in this approach.
>
> `LeakSUSK(I)` The time-independent static key allows constructing any key for epoch I as long as an update token for an epoch t has been published. The resulting attacks on `fs`, `pcs`, `dker_f` and `dker_b` are trivial, because they are independent of the assumed leak.

Individual token (separate) There is one more special attack vector for the separate-key variation of the individual token approach:

> **`LeakUSK(I,x) for any epoch` x `before` t** In this variation, the static key can be extracted from $usk_{ID,x}$ for any x. The leak of previously existing $usk_{ID,x}$ produces trivial attacks in the lemmas for `fs` and `dker_f`; when it is excluded, TAMARIN finds a non-trivial attack where a leak from a later epoch is used to derive the user key of an earlier one.

Universal token For the universal-token approach, additional trivial attacks need to be excluded in some lemmas before TAMARIN finds non-trivial attacks in the given model.

> **`LeakUpdVal(R,t)` $\wedge$ `LeakMSK(x)`** $\forall x > 0$ For `dker_b`, which is achieved if earlier `usk`-leaks for a user `I` do not harm the security of $usk_{I,t}$, a trivial attack is computing the key from an earlier or later *msk*-leak and any user's leaked token for the targeted epoch `t`.

7.2.4. Non-trivial attacks

The following attacks are considered non-trivial, as explained below. A non-trivial attack means that the security property is not (unconditionally) achieved, and the assessments are recorded in Table 7.2 accordingly.

Individual Token, "separate" approach

When the static key and the update token are not re-randomized in computing $usk_{ID,t}$, the adversary can directly derive the static key from it if leaked (which is assumed in the lemmas for `fs`, `pcs`, `dker_f` and `dker_b`). Since the update tokens are public, an attacker with a static key can derive a user key for any epoch in which an update token was published[3]. Thus, the key or message from an earlier (`fs`, `dker_f`) or later epoch (`pcs`, `dker_b`) is compromised, too, if the key was updated for it. This contradicts the stated lemmas and the security properties are not achieved.

As Table 7.2 indicates, the variant where static key and token are re-randomized to yield the user secret is secure against these attacks; here, leaking the user secret does not allow an adversary to learn the static key and the same attack is not possible.

Universal Token Approach

In the universal token approach, there are two kinds of attacks respectively undermining:

> **`fs`, `pcs`, `dker_f` and `dker_b`:** Although the leak of one user secret key does not directly harm security (as it does for the separate-key variation of the individual token approach), the leak can be exploited if a suitable update token for any user is leaked as well. Therefore, this attack is conditional but not trivial.

[3] In the TAMARIN counterexamples, the published tokens are for (yet) non-existent keys. That is an implausible scenario; however, the attack does not depend on the key having been generated after the token publication, so the attack itself is plausible. Restrictions to enforce token distribution only for existing keys make the model more complex; no suitable restriction was found.

For `fs` and `dker_f`, which would be achieved if a later leak did not impact the security of an earlier key/message encrypted for the same user, the attack is possible when an update for the later epoch is also leaked. Then the attacker can downgrade the updated key using the token.

For `pcs` and `dker_b`, which would be achieved if an earlier leak did not impact the security of a later key/message encrypted for the same user, the attack is possible when an update for the later epoch is also leaked. Then the attacker can update the key using the token. The leaked update token may be labeled with any identity; since it is the same for all members, the attacker does not need to compromise the targeted user's token.

Collusion Resistance: Even when no user key or update token is generated for an identity ID in an epoch t, it is possible to decrypt a respectively encrypted message or derive $usk_{ID,t}$ if both a user key for an earlier or later session and at least one update token for t (that was generated for any identity, not necessarily ID) are leaked. Therefore, this attack is conditional but not trivial, and the universal token approach is not collusion resistant.

As an additional minimality check, `lemma collusion_resistance_proof` shows that the key/message for a revoked user remain unknown to an attacker if neither any msk is ever leaked nor any user's token for epochs t or $t + 1$.

7.3. Evaluation

Even though the models abstract the precise mathematics of the underlying schemes, they capture the characteristics well enough to replicate findings from the literature and offer additional insight. The following sections discuss:

- the drawbacks and strengths of each approach
- the findings on DEKR in the individual-token approaches, which align with previous research
- the findings on Forward Security and Post-Compromise Security in the universal-token approach, which disagree with previous research
- Collusion Resistance and the revocation effect
- how to reconcile the notions of DKER and Forward-/Post-Compromise Security
- how to mitigate attacks from forward/backward effects
- the benefits and drawbacks of a manual vs. an automated security analysis
- limitations of the presented models

7.3.1. Approaches' drawbacks and strengths

The drawbacks and strengths are implicit in the specific trivial attacks found for each scheme, which highlight the comparative (but not absolute) weaknesses of each scheme. For example, a secure channel for update distribution is paramount in the key renewal approach (where the leak of an update value appears as a specific trivial attack), whereas the individual-token approach allows publishing the update value with no direct impact. Conversely, the need to keep a static key secure only appears in the individual-token approach.

While leaking the update token hurts the security of keys and messages more in the universal token approach, leaking the master secret for any given epoch is less impactful than in other schemes: Supposing the TTP can re-gain control after a compromise, the *msk* (and thus all keys) can be "healed" by generating a fresh update token and the system can move on from the leak. In all other mechanisms, an *msk* leak is fatal and the system needs to be set up from scratch.

7.3.2. Decryption Key Exposure attacks in the individual-token approaches

Tamarin finds the Decryption Key Exposure attacks against the individual-token approach when there is no re-randomization (which resembles the approach in [BGK08]) and their absence when there is re-randomization, reproducing the results from the literature [SE13b].

The analysis in this work goes beyond the literature findings in formulating distinguishing notions for forward and backwards security against Decryption Key exposure. While no noteworthy difference was observed in the analysis, the distinction may prove useful for future proposals.

7.3.3. Forward Security and Post-Compromise Security in the universal-token approach

The Tamarin analysis finds non-trivial attacks against Forward- and Post-Compromise Security in the universal token approach. However, the underlying literature for this approach, [Gug20], describes Key Updatable Signature Schemes (KUSS) with a universal update token and finds them to be Forward- and Post-Compromise secure.

We argue that 1. our findings apply to KUSS-schemes and 2. the discrepancy lies in the definition of an attack:

1. Even though the model represents encryption rather than signature schemes, the attacks found in Tamarin can be seamlessly transferred: The effect of leaking a token with which to update a signature key is the same as for the encryption key, and this is the most prominent attack vector against the revocation scheme. Also, the Tamarin model abstracts from an additional, accumulative value necessary for signature verification in three out of four of the mechanisms described in [Gug20]. This value records the accumulated update tokens across epochs. However, the attacks remain possible despite the use of this value, as it does not change the way in which tokens are distributed and applied in each epoch.

2. The definition of Forward Security in [Gug20] states:

   > "An adversary compromising some usk and [the accumulated update value for epoch e^*] in some epoch e^* does not gain any advantage in forging a signature for previous epochs $e < e*$. [...]" (see [Gug20], page 49).

> As [Gug20] asserts without further explanation that the mechanisms in question achieve Forward Security (ibid., page 76), the *usk* leak is evidently not considered an advantage - perhaps because an additional leak is necessary to exploit it. The respective paragraph informally explains that Forward Security is achieved against former members who re-join the communication group if they do no get enough information upon re-entry to sign for previous epochs in which they were not members.
>
> However, our work considers any attack possibility for an adversary that arises from the key compromise in epoch e^* an advantage (even if the attacker is a legitimate member in epoch e^*). Therefore, we conclude that the universal token approach does not achieve Forward Security.

Along the same lines, the TAMARIN analysis and the analysis in [Gug20] come to different conclusions regarding Post-Compromise Security: The attack mechanism holds for the presented schemes, but while this work interprets it as evidence that the property is not achieved, [Gug20] asserts otherwise, explaining that the epoch transition protects new keys against former members (but not discussing other attackers).

7.3.4. Collusion Resistance and revocation effect

The protection against former and future members is also implicit in the notion of Collusion Resistance. TAMARIN finds a non-trivial attack against the universal token approach, which requires at least two leaks. While this can be a relatively high bar to clear for an attacker, in this case it is not: Key revocation sometimes serves to exclude malignant or compromised users, whose latest user key is known to the attacker (possibly because the user herself is the attacker). So if, for example, a revoked, malignant user colludes with a non-revoked user, she can apply their update token to her own key. Thus, the attack is non-trivial and realistic.

Achieving Collusion Resistance is also a useful sanity check for how effective an approach is as a revocation mechanism: If, without using the master secret key, a user key can be obtained for epochs for which the users should not possess one, the revocation is only effective under stronger assumptions than if a master secret key is needed to obtain non-existing keys. A realistic collusion scenario thus implies a weaker revocation effect.

7.3.5. Reconciling Decryption Key Exposure Resistance and Forward-/Post-Compromise Security

The separate individual token approach and the universal token approach are vulnerable to Decryption Key exposure in either direction. Both approaches are also not Forward- or Post-Compromise secure. Within each model, the attacks that are found against `dker_f` and against `fs` exhibit the same structure, i.e. the leak that is assumed in both notions is exploited in the same way to yield the user secret of another epoch (`dker_f`) or the decrypted message (`fs`). This indicates that the notions are very similar. An attacker who can derive a secret key can also decrypt messages with it. Therefore, it is intuitive that `fs` implies `dker_f`: If the attacker can not decrypt a cipher, they clearly do not know the corresponding user secret key. The reverse is not necessarily true, because an attacker who has no user secret key may still find other ways to decrypt the cipher. However, the considered models allow no such attack, so within their scope, both notions are equivalent.

In the same way, `dker_b` and `pcs` are parallel notions that yield structurally similar attacks, of which `pcs` intuitively implies `dker_b`, and which within the scope of the considered models are equivalent (although they may not be in other contexts).

That `dker_f` and `dker_b` are so similar to `fs` and `pcs`, respectively, also highlights the value of splitting the notion of decryption key exposure, which in the literature is treated as a single property, in two notions that separately consider the impact for future and for former keys.

7.3.6. Mitigating attacks using forward/backward effects

In both the separate individual token approach and the universal token approach, a user key leak from one epoch can be directly exploited by applying an update token to obtain a key for earlier or later epochs.

For the individual token approach, it is clear that the attacks can be mitigated because the re-randomized version does not observe them.

For the universal token approach, Forward Security would be achieved if updating a key was mathematically irreversible, because applying the update token to obtain an earlier version of the key is not an intended use of the token; so this functionality is not needed. However, such a re-randomization might interfere with the decryption (signature verification) process if the connection to *msk* can not be shown from the re-randomized key anymore; so it is not clear that a suitable, unidirectional function exists.

Post-Compromise Security would afford an entirely different update logic: In the current form, applying the update token directly to the user key is the intended use of the token, so compromising the two values seems to be an almost trivial attack. To achieve Post-Compromise Security, the user key must not yield enough information to update it directly. Whether this condition is reconcilable with the universal token approach remains an open question.

7.3.7. Manual vs. automated analysis

As the attack against a mechanism in [BGK08] that was discovered in [SE13b] and the attacks against the universal token approach described in Section 7.2 indicate, revocation mechanisms in IBC can have attack vectors that a manual analysis underestimates or overlooks. A computer-aided analysis may be less prone to error because the presentation and results are as clear and easy to understand as the models in manual analyses, but the evaluation of the security notions automatically uncovers also less obvious attacks. Once a blue-print is established, the models are also easily adaptable and the comparison of various mechanisms is very straightforward.

On the downside, the analysis is only possible on a more abstract level and mathematical effects from specific key derivation functions can be hard to capture in the model. When the results from a computer-aided analysis are used to argue for a scheme's security, it therefore needs to be addressed how faithfully the underlying functions and processes are represented, so that readers understand the explanatory power of the model well.

The best use of computer-aided analysis in identity-based revocation therefore lies in understanding and comparing relatively abstract approaches to learn how various features and properties influence the security. Specific instantiations of an approach still require a careful analysis of the underlying functions that are abstracted in the model.

In a similar way, other revocation- and epoch-based key management systems may benefit from computer-aided analysis and the models developed and exemplified in Chapter 5 and Chapter 6.

7.3.8. Limitations

The security assessment and comparison have the following limitations:

`rule update_msk` The models are not perfectly comparable: One includes `rule update_msk` and the others do not. However, the observed effects from the *msk* update are impossible in the other approaches. Therefore, it is expected that the comparison yields the same results as a perfect one.

function `msk_der(·)` in universal token approach As stated in Section 6.5.5, the attack vector for computing earlier/later master secrets through update tokens or vice versa are excluded for complexity reduction. The precise strengths and weaknesses of updating the master key material are therefore not reflected in the analysis. Also, some mechanisms in [Gug20] rely on an extra value for verification and signing. This value was omitted because it does not appear in all cases, but a more detailed analysis is warranted.

7.4. Characteristics, revisited

This chapter answers **RQ 4)** ("Does the formal verification confirm previous results? Does it yield any new insights?"). Table 7.4 shows the completion of Table 4.1 based on the results from the automated analysis.

Except for [Gug20], the results are congruent with all previous findings. Because the notion of Decryption (Signing) Key Exposure (one row in Table 4.1) is split in two separate notions for the analysis, the table has an extra row. The notions for Forward- and Post-Compromise security, while slightly different than the key exposure notions, add little new insight per scheme. If the mathematics of encryption/signing are as abstract as in the presented models, one of the two sets of notions seems to suffice. Collusion Resistance can be inferred for all schemes except the ones presented in [Gug20]. There, a non-trivial attack is found, which indicates that Collusion Resistance should be considered in a comprehensive security analysis.

Table 7.4.: Characteristics and properties of identity-based cryptography schemes as presented in Table 4.1: Again, each entry indicates whether the publication *(column)* claims that their mechanism does (✓) or does not (✗) have a property *(row)*. Empty cells indicate that the property is not definitively addressed in the publication itself or any related work item. The characteristics established or re-evaluated in this work are encircled (✓ or ✗); previous assessments (where they exist) are denoted in the same cell for comparison.
There is an extra row compared to Table 4.1 because decryption key exposure is split in two notions (forward and backward).
References in the table: $*$ = [SE13b] † = [ML20]

Properties	[BF01]	[BGK08]	[LV09]	[Che+12]	[SE13b]	[ML20]	[Liu+16]	[XWW20]	[Gug20]
Enc/Sig	E	E	E	E	E	E	S	S	S
strategy	re-key	token	token	token	token	token	token	token	token
Gen	✓	✗			✓	✓			✓
static sk	✗	✓	✓	✓	✓	✓	✓	✓	✗
re-rand.		✗*	✗*	✗*	✓	✓	✓	✓	✗
upd. size	$\mathcal{O}(n-r)^\dagger$	$\mathcal{O}(r \log \frac{n}{r})^\dagger$	$\mathcal{O}(r \log \frac{n}{r})^\dagger$	$\mathcal{O}(r \log \frac{n}{r})^\dagger$	$\mathcal{O}(r \log \frac{n}{r})^\dagger$	$\mathcal{O}(r \log \frac{n}{r})$	$\mathcal{O}(r \log \frac{n}{r})$	$\mathcal{O}(\log r)$	$\mathcal{O}(1)$
bin. tree	✗	✓	✓	✓	✓	✓	✓	✓	✓
upd. dist.	$\mathcal{O}(n-r)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n-r))$
D/SKER B.	✓*✓	✗*✗	✗*✗	✗†✗	✓✓	✓✓	✓✓	✓✓	✗
D/SKER F.	✓	✗	✗	✗	✓	✓	✓	✓	✗
Forw. Sec.	✓	✗	✗	✗	✓	✓	✓	✓	✓✗
Post-C. Sec.	✓	✗	✗	✗	✓	✓	✓	✓	✓✗
Coll. res.	✓✓	✓✓	✓	✓	✓✓	✓	✓	✓	✗

8. Conclusion

Identity-based cryptography (IBC) reduces the key management overhead compared to certificate-based methods because it allows the derivation of a user's public key from their identity. Using revocation lists to verify key validity would negate these benefits. Efficient revocation of identity-based keys has been addressed in several works, but they only propose individual mechanisms.

This work goes beyond discussing a single approach and offers a more general perspective on IBC-revocation, finding the following:

- Fundamentally, cryptographic key revocation methods for asymmetric keys can be
  a) either explicit or implicit, depending on whether or not users need to check key validity before use
  b) either direct or indirect, depending on whether or not a key revocation affects only the keys of compromised entities or also others

  All identity-based approaches fall into the category "indirect and implicit" because key users do not need to check key validity and key revocation affects also non-revoked keys.

- Three classes of IBC-revocation appear in related work:
  1. Key renewal: Entirely new keys are computed for all users and it requires a separate private channel to the TTP for each user. Therefore, the computation- and distribution complexity is linear in the number of users.
  2. Updates with individual tokens: They have constant distribution complexity because the updates can be published. The computation complexity is logarithmic if users can hold several, hierarchically structured keys; otherwise, it is linear.
  3. Updates with a universal token: They imply minimal computation load for the TTP but require private channels for the update distribution. If users can hold several keys for this channel, the distribution complexity is logarithmic, otherwise it is linear.

- Since [SE13b] introduced the security notion of Decryption Key Exposure Resistance, most of the recent literature proposing IBC-revocation schemes considers it. This notion focuses on how key compromises in one epoch effect keys of other epochs. One recent scheme [Gug20] considers Forward- and Post-Compromise Security instead, which aim at the effect of leaks on messages rather than keys.

  These notions are reconciled in this work with the introduction of *Decryption Key Exposure Resistance Forward* and *-Backward*. In the Tamarin analysis,
  - if one of the forward (backward) notions is violated, so is the other.
  - the attacks against both forward (backward) notions are structurally the same.

In our analysis, the distinction between forward and backward notions is largely inconsequential: The considered revocation approaches achieve either both or neither. However, future analyses might benefit from working with the more precise variants of Decryption Key Exposure in the same way as literature consistently distinguishes Forward- and Post-Compromise Security.

- Key renewal and individual token updates achieve Collusion Resistance. The universal token approach does not. Collusion Resistance is introduced as a new security notion to understand whether users can combine their secret knowledge to obtain information they must not have individually.

The main contribution of this work is a framework for the automated analysis of IBC-revocation approaches, detailing:

a) a state diagram for IBC-revocation

b) security goals

c) dependencies and compromise options exploitable by an adversary

With a proof-of-concept implementation of the formalization in Tamarin, this work further contributes a tangible comparison between the considered IBC-revocation approaches:

The proof-of-concept implementation shows that the formalization is adaptable to all identified revocation approaches. It even yields two more granular models for individual token updates, in which the structure of the decryption keys reflects the concept of re-randomization used in some (but not all) instantiations of this approach.

The results obtained from running the four models in Tamarin are only partially in line with previous findings (where they exist):

- For the individual token approaches, the formalization confirms the attacks against mechanisms that do not re-randomize keys when updating and that re-randomization mitigates this problem.

- For the universal token approach, the claimed Forward- and Post-Compromise Security are not confirmed; though the previous work recognizes the same attack vectors but deems them negligible, the formalization in this work concludes that they impair the security.

Besides the comparison with existing results, the formalization and implementations also delivers new insight by checking the whole range of security notions for all classes rather than different notions for different classes. The results show

a) that Decryption Key Exposure Resistance is achieved when Forward- and Post-Compromise Security are achieved (and vice versa), thus reconciling these notions,

b) that Decryption Key Exposure Resistance can be evaluated for effects on past and on future keys separately, and that attacks in both directions exist, and

c) that Collusion Resistance is not achieved for all classes, which implies that it should be included in a standard security analysis for IBC-revocation.

With the formalization and its implementation, this work contributes a tangible comparison between the approaches. Automated analysis is therefore a valuable addition to (often tedious) manual analysis. The danger of incorrect modeling still exists, and in incorrect models, possible attacks may go unnoticed; however, the analysis process is different from manual analysis and can therefore automatically produce insights that may be overlooked more easily in a manual proof. The analysis remains high-level as the tool can not model certain mathematical connections in more depth; however, to understand the protocol-level implications of update mechanisms in key management, the models are well-suited.

Future work

The models implemented for this work offer a comprehensive overview of IBC-revocation approaches (with encryption as the main focus), but the formalization and the blueprint are not restricted to this scenario. There are several obvious variations of the proposed models, for example implementing a signature scheme or modeling the master secret key dependencies for the universal token approach in greater detail.

Should new ideas for other IBC-revocation approaches emerge, automated analysis can be useful in the design process to recognize and mitigate security flaws early on, and the Tamarin blueprint for the model can be used as is to model even a scheme that is yet unknown. Since the European Telecommunications Standards Institute (ETSI) recommends IBC for use cases such as the Internet of Things and "public safety and mission critical applications which require secure one-to-one, group and broadcast communications" [Eur22], further research in this area can be expected.

The blueprint itself might evolve, too: There are currently research efforts to improve how Tamarin handles loops and recursions[1], which might allow running a less restricted version of the universal token model (for example). Improvements with respect to the mathematical dependencies that Tamarin (and trace-based provers in general) can handle might also make it possible to implement identity-based cryptography and -revocation in more detail.

Looking beyond IBC, the underlying update logic of the formalization appears in other areas of cryptography, as well: The formalization lends itself to be adapted for ratcheting or other key management approaches in which keys are updated throughout sequential epochs (see, for example, the proposal for "Key-updatable public-key encryption" [Ana+20] and the follow-up work [Eat+22].) The repeated references to the Internet of Things as a use case for key-updatable public key mechanisms [Gug20; Ana+20] is an indicator that this research area may draw more interest in the future. If that is the case, the results from this work may prove a valuable support for understanding the security of such mechanisms.

[1] by Felix Linker, PhD student with ETH Zurich's Information Security Group: "Currently, I work on applying cyclic proof rules to Tamarin such that it can handle looping protocols and recursive data structures better. More soon!" (cited from `https://felixlinker.de/works/`)

A. Blueprint for Tamarin code that models an encryption process with key revocation

The following code blueprint is a translation of the formalization presented in Chapter 5 to Tamarin code in accordance with the explanation given in Chapter 6.

Portions highlighted in yellow indicate placeholders: For the blueprint to run in Tamarin, the placeholders need to be substituted with code implementing the corresponding parts of the model.

```
1  theory %theory name
2  begin
3
4  builtins: asymmetric-encryption, hashing %further built-in(s)
5  functions: ibenc/4, ibdec/2
       %further function(s) %adjust arity if necessary
6  equations: %the encryption/decryption equation of each model
7      % and any further equation(s)
8
9  // Setup of the master key material and start of first epoch
10     rule setup:
11     let
12     mpk = pk(~msk)
13     in
14     [ Fr(~msk)           // msk is a fresh random value
15     , Fr(~t)             // first epoch identifier is a fresh
           random value
16     ]
17     --[ SetupOnlyOnce(), Is_Epoch(~t) ]-> // system is set up
           only once, which happens in epoch t
18     [ !MSK(~msk, ~t)
19     , !MPK(mpk, ~t)
20     , !Epoch(~t)
21     , Out(mpk)           // mpk and epoch identifier are public
           values
22     , Out(~t) ]
23
```

```
24 // Generation of key material for a user I from scratch in an
       epoch t
25 rule generate_usk:
26     let
27     %formula for the computation of usk
28     in
29     [ !Epoch(~t)              // current epoch is t
30     , !MSK(msk, ~t) ]         // and msk exists
31     --[ GenUSK($I), Is_Epoch(~t), Exists_Key($I, usk, ~t) ]->
32     [ !USK(usk, $I, ~t) %byproduct facts]
          // usk for I and epoch t exists
33
34 //  Advance the system to the next epoch
35     rule advance_epoch:
36     [ !Epoch(~t1)             // If epoch t1 exists,
37     , Fr(~t2)                 // draw a fresh random value.
38     ]
39     --[ Epoch_ended(~t1), Consecutive(~t1, ~t2) ]->
40     [ !Epoch(~t2)                  // epoch t2 exists
41     , !PreviousEpoch(~t1,~t2)      // epoch t2 directly follows
          epoch t1
42     , Out(~t2)]                    // epoch identifier t2 is public
          knowledge
43
44 //Update the master key material
45 rule update_msk:
46     let
47     msk_new = %formula for msk update
48     mpk_new = pk(msk_new)
49     in
50     [ !Epoch(~t_new)          // If epoch t1 exists
51     , !PreviousEpoch(~t_old, ~t_new) // as a follower of epoch
          t_old,
52     , !MSK(msk_old, ~t_old) // and msk_old is the master secret
          from the previous epoch,
53     , Fr(~upd_val) ]     // choose a fresh value to apply as an
          update.
54     --[ Update_MSK(msk_new, ~t_new), Is_Epoch(~t_new) ]->
55     [ !MSK(msk_new, ~t_new)  //msk_new exists
56     , !MPK(mpk_new, ~t_new)      //mpk_new exists
57     , Out(mpk_new)               //mpk_new is public
58
59
60 // Create and distribute an update value to non-revoked user I
61 rule distribute_token:
62     let
63     update_value =
          %formula for the computation of the update value
64     in
65     [ !Epoch(~t_new)          // If epoch t1 exists
66     , !MSK(msk, ~t_new)          // and msk for this epoch does
67     ]
68     --[ Is_Epoch(~t_new), Create_Update($I, ~t_new) ]->
```

```
69      [ !Update_Value($I, ~t_new, update_value) //create an update
            value for user I
70      %other output for token distribution
71      ]
72
73  // Update the user key
74  rule update_usk:
75      let
76      usk_new = %formula for the computation of the usk update
77      in
78      [ !Epoch(~t_new)             // If epoch t_new exists
79      , !PreviousEpoch(~t_old,~t_new) // as a follower of epoch
            t_old,
80      , !Update_Value($I, ~t_new, upd_val) // and there is a
            current update token for I
81      , !USK(usk_old, $I, ~t_old)]    // who had a usk in the epoch
            before,
82      --[ Update_USK($I, ~t_new), Exists_Key($I, usk_new, ~t_new),
            Is_Epoch(~t_new) ]->
83      [ !USK(usk_new, $I, ~t_new)]    // then I computes an updated
            usk.
84
85  // Encrypt a message for a user ...
86  rule encrypt_message:
87      let
88      %formula for the encryption of message m
89      in
90      [ !MPK(mpk, ~t)              // using mpk,
91      , Fr(~m)                     // a fresh random (=unique)
            message m,
92      , !Epoch(~t)                 // for epoch t.
93      ]
94      --[ Used($I, ~t), Sent(~m, $I), Is_Epoch(~t)]->
95      [ Out(cipher)           // The cipher is sent over an
            insecure channel (= public knowledge).
96      ]
97
98  // Decrypt a message ...
99  rule decrypt_message:
100     let
101     %pattern match for the usk
102     %pattern match for the cipher
103     %computation formula for decrypted message
104     in
105     [ In(cipher)                 // (that comes from an insecure
            channel)
106     , !MPK(pk(msk), ~t)          // using mpk
107     , !USK(usk, $I, ~t)          // and the user I's usk.
108     ]
109     --[ Receive(m_dec), Read($I, m_dec, ~t) ]->
110     [ ]
111
112 %any additional rules
```

```
113
114 // Key  Leaks as defined in the adversary model
115 rule leak_msk:
116     [ !MSK(msk,~t) ]
117     --[ LeakMSK() ]->
118     [ Out(msk) ]
119
120 rule leak_usk:
121     [ !USK(usk, $I, ~t) ]
122     --[ LeakUSK($I, ~t) ]->
123     [ Out(usk) ]
124
125 rule leak_upd_val:
126     [ !Update_Value($I, ~t, update_value_I) ]
127     --[ LeakUpdVal($I, ~t) ]->
128     [ Out(update_value_I) ]
129
130 %leak for other secret values
131
132
133 // Management-level constraints for the trace construction rules
134 // ============================================================
135
136 restriction SetupOnlyOnce: "All #i #j. SetupOnlyOnce()@ i &
       SetupOnlyOnce()@ j ==> #i = #j"
137
138 restriction End_Epoch_Once:
139 "All t #i #j. Epoch_ended(t)@ i & Epoch_ended(t)@ j ==> #i = #j"
140
141 restriction End_of_Epoch_holds:
142 "All t #i #j. Epoch_ended(t)@ i & Is_Epoch(t)@ j ==> #j < #i"
143
144 restriction USK_Gen_or_Upd_only_once_per_epoch:
145 "All I usk1 usk2 t #i #j. Exists_Key(I, usk1, t)@ i &
       Exists_Key(I, usk2, t)@ j ==> #i = #j"
146
147 restriction GenUSK_only_if_not_member_in_previous_epoch:
148 "All I t1 t2 #i #j. Consecutive(t1, t2)@ #j & GenUSK(I)@ #i &
       Is_Epoch(t2) @ #i ==> not(Ex usk #k. Exists_Key(I, usk, t1)@
       #k)"
149
150 restriction CreateUpdate_and_GenUSK_not_in_the_same_epoch:
151 "All I t1 t2 #i #j. Consecutive(t1, t2)@ #j & Create_Update(I,
       t2)@ #i
152 ==> not (Ex #k. GenUSK(I)@ #k & Is_Epoch(t2)@ #k)"
153
154 restriction CreateUpdate_only_once_per_epoch:
155 "All t I #i #j. Create_Update(I, t)@ #i & Create_Update(I, t)@ #j
       ==> #i = #j"
156
157 restriction Update_MSK_only_once_per_epoch:
158 "All t msk1 msk2 #i #j. Update_MSK(msk1, t)@ #i &
       Update_MSK(msk2, t)@ #j ==> #i = #j"
```

```
159
160 restriction Leak_USK_only_once_per_epoch:
161 "All I t #i #j. LeakUSK(I, t)@ i & LeakUSK(I, t)@ j ==> #i = #j"
162
163 % additional restrictions
164
165 % sources lemmas
166
167 // Sanity check lemmas
168 // ====================
169 lemma can_receive: exists-trace
170     /* There exists a session in which a user can decrypt a
        message. */
171     "Ex m #j. Receive(m)@ #j
172     & not (Ex #i t N. LeakUSK(N,t)@ #i)
173     & not (Ex #i t. LeakMSK(t)@ #i)
174     & not (Ex I t #l. LeakUpdVal(I,t)@ #l)
175     %no other leak
176     "
177
178 lemma can_receive_after_update: exists-trace
179     /* There exists a session in which a user can decrypt a
        message after their key was updated. */
180     "Ex m I t t2 #j #k #l. Read(I, m, t2)@ #j
181     & Consecutive(t,t2) @ #k
182     & Update_USK(I, t2) @ #l
183     & k < j
184     & not (Ex #i t N. LeakUSK(N,t)@ #i)
185     & not (Ex #i t. LeakMSK(t)@ #i)
186     & not (Ex I t #l. LeakUpdVal(I,t)@ #l)
187     %no other leak
188     "
189
190 lemma two_users_can_have_keys_in_same_epoch: exists-trace
191     "Ex I1 I2 usk1 usk2 t #j #i. not(I1=I2)
192     & Exists_Key(I1, usk1, t)@ #i
193     & Exists_Key(I2, usk2, t)@ #j
194     "
195
196 lemma minimal_example: exists-trace
197     "Ex
198     t1 t2 t3 t4
199     #r1 #r2 #r3
200     A B C  D
201     m1 m2 m3 m4
202     #ga #gc #gd
203     #s1 #s2 #s31 #s32 #s4
204     #rm1 #rm2 #rm31 #rm4
205     #ua2 #ub2 #ub3 #uc3 #ub4 #uc4.
206     Consecutive(t1,t2)@ #r1 & Consecutive(t2,t3)@ #r2 &
        Consecutive(t3,t4)@ #r3 & #r1 < #r2 & #r2 < #r3
207     & (All t5 t6 #r4. Consecutive(t5,t6)@ #r4 ==> ( #r4 = #r1 |
        #r4 = #r2 | #r4 = #r3))
```

```
208     & GenUSK(A)@ #ga & Sent(m1, B)@ #s1 & Read(B, m1, t1)@ #rm1
209     & Update_USK(A, t2)@ #ua2 & Update_USK(B, t2)@ #ub2 &
            GenUSK(C)@ #gc & Sent(m2, C)@ #s2 & Read(C, m2, t2)@ #rm2
            & #r1 < #gc & #gc < #r2
210     & Update_USK(B, t3)@ #ub3 & Update_USK(C, t3)@ #uc3 & not (Ex
            #ua3 uskA_3. Exists_Key(A, uskA_3, t3)@ #ua3) & Sent(m3,
            C)@ #s31 & Read(C, m3, t3)@ #rm31 & Used(A, t3)@ #s32 &
            not (Ex m #rm33. Read(A, m, t3)@ #rm33)
211     & Update_USK(B, t4)@ #ub4 & Update_USK(C, t4)@ #uc4 &
            GenUSK(D)@ #gd & Sent(m4, D)@ #s4 & Read(D, m4, t4)@ #rm4
            & #r3 < #gd
212     & not (Ex id t #leakUSK. LeakUSK(id, t)@ #leakUSK) & not (Ex
            #i t. LeakMSK(t)@ #i) & not (Ex I t #l. LeakUpdVal(I,t)@
            #l)
213     %no other leak
214     "
215
216 // Security properties
217 // ===================
218
219 /*
220 * Forward security:
221 * Messages encrypted for user I and an epoch t remain
      confidential,
222 * even if I's secret key for later epoch t+x is leaked
223 * (unless there is a trivial attack w.r.t. epoch t).
224 */
225 lemma forward_security:
226     "All m I t1 t2 #i #j #k #o.
227     Sent(m,I) @ #i
228     & Is_Epoch(t1) @ #i
229     & Epoch_ended(t1) @ #j
230     & LeakUSK(I,t2) @ #k
231     & Is_Epoch(t2) @ #o
232     & #j < #o
233     ==> not(Ex #l. K(m)@ #l)
234     | (Ex #l. LeakUSK(I, t1)@ #l)
235     | (Ex #l. LeakMSK(t1)@ #l)
236     | (Ex #l. LeakUpdVal(I,t1)@ #l)
237     %other trivial leak
238     "
239
240 /*
241 * Post-Compromise Security:
242 * Messages encrypted for user I and an epoch t remain
      confidential,
243 * even if I's secret key for an earlier epoch t-x is or was leaked
244 * (unless there is a trivial attack w.r.t. epoch t).
245 */
246 lemma post_compromise_security:
247     "All m I t1 t2 #i #j #k.
248     LeakUSK(I,t1) @ #i
249     & Epoch_ended(t1) @ #j
```

```
250     & Sent(m,I) @ #k
251     & Is_Epoch(t2)@ #k
252     & #j < #k
253     ==> not(Ex #l. K(m) @ #l)
254     | (Ex #l. LeakUSK(I, t2)@ #l)
255     | (Ex #l. LeakMSK(t2)@ #l)
256     | (Ex #l. LeakUpdVal(I,t2)@ #l)
257     %other trivial leak
258     "
259
260 /*
261 * DKER forward:
262 * A user I's secret key for an epoch t remains secret,
263 * even if I's secret key for later epoch t+x is leaked
264 * (unless there is a trivial attack w.r.t. epoch t).
265 */
266 lemma decryption_key_exposure_resistance_forward:
267     "All I t1 t2 usk1 usk2 #i #j #k.
268     Exists_Key(I, usk1, t1)@ #i
269     & Is_Epoch(t1)@ #i
270     & Exists_Key(I, usk2, t2)@#j
271     & #i < #j
272     & LeakUSK(I, t2)@ #k
273     ==> not (Ex #l. K(usk1)@ #l)
274     | (Ex #l. LeakUSK(I, t1)@ #l)
275     | (Ex #l. LeakMSK(t1)@ #l)
276     | (Ex #l. LeakUpdVal(I,t1)@ #l)
277     %other trivial leak
278     "
279
280 /*
281 * DKER backward:
282 * A user I's secret key for an epoch t remains secret,
283 * even if I's secret key for an earlier epoch t-x is or was leaked
284 * (unless there is a trivial attack on epoch t).
285 */
286 lemma decryption_key_exposure_resistance_backward:
287     "All I t1 t2 usk1 usk2 #i #j #k.
288     Exists_Key(I, usk1, t1)@ #i
289     & Is_Epoch(t1)@ #i
290     & Exists_Key(I, usk2, t2)@#j
291     & #i < #j
292     & LeakUSK(I, t1)@ #k
293     ==> not (Ex #l. K(usk2)@ #l)
294     | (Ex #l. LeakUSK(I, t2)@ #l)
295     | (Ex #l. LeakMSK(t2)@ #l)
296     | (Ex #l. LeakUpdVal(I,t2)@ #l)
297     %other trivial leak
298     "
299
300 /*
301 * Collusion Resistance:
302 * If no update and no fresh key were generated for user I in an
```

```
303     epoch t,
304 * the adversary can not know a message m encrypted for I
305 * (unless the master secret for t is leaked).
306 */
307 lemma collusion_resistance:
308     "All I m t #i.
309     Sent(m, I)@ #i
310     & Is_Epoch(t)@ #i
311     & not (Ex #j. Create_Update(I, t)@ #j)
312     & not (Ex #j. GenUSK(I)@ #j & Is_Epoch(t)@ #j)
313     ==> not(Ex #l. K(m)@ #l)
314     | (Ex #l. LeakMSK(t)@ #l)
315     "
316
317 // minimal assumption checks
318 // ========================
319
320 // Forward Security
321 //-----------------
322
323 // Leaking usk of the target epoch yields a trivial attack.
324 lemma forward_security_uskleak_attack:
325     "All m I t1 t2 #i #j #k #o.
326     Sent(m,I) @ #i
327     & Is_Epoch(t1) @ #i
328     & Epoch_ended(t1) @ #j
329     & LeakUSK(I,t2) @ #k
330     & Is_Epoch(t2) @ #o
331     & #j < #o
332     ==> not(Ex #l. K(m)@ #l)
333     // | (Ex #l. LeakUSK(I, t1)@ #l)
334     | (Ex #l. LeakMSK(t1)@ #l)
335     | (Ex #l. LeakUpdVal(I,t1)@ #l)
336     %other trivial leak
337     "
338
339 // Leaking msk of the target epoch yields a trivial attack.
340 lemma forward_security_mskleak_attack:
341     "All m I t1 t2 #i #j #k #o.
342     Sent(m,I) @ #i
343     & Is_Epoch(t1) @ #i
344     & Epoch_ended(t1) @ #j
345     & LeakUSK(I,t2) @ #k
346     & Is_Epoch(t2) @ #o
347     & #j < #o
348     ==> not(Ex #l. K(m)@ #l)
349     | (Ex #l. LeakUSK(I, t1)@ #l)
350     //| (Ex #l. LeakMSK(t1)@ #l)
351     | (Ex #l. LeakUpdVal(I,t1)@ #l)
352     %other trivial leak
353     "
354
```

```
355 // Leaking the update value for the target epoch yields a trivial
        attack.
356 lemma forward_security_tokenleak_attack:
357     "All m I t1 t2 #i #j #k #o.
358     Sent(m,I) @ #i
359     & Is_Epoch(t1) @ #i
360     & Epoch_ended(t1) @ #j
361     & LeakUSK(I,t2) @ #k
362     & Is_Epoch(t2) @ #o
363     & #j < #o
364     ==> not(Ex #l. K(m)@ #l)
365     | (Ex #l. LeakUSK(I, t1)@ #l)
366     | (Ex #l. LeakMSK(t1)@ #l)
367     //| (Ex #l. LeakUpdVal(I,t1)@ #l)
368     %other trivial leak
369     "
370
371 // There is another trivial attack.
372 lemma forward_security_suskleak_attack:
373     "All m I t1 t2 #i #j #k #o.
374     Sent(m,I) @ #i
375     & Is_Epoch(t1) @ #i
376     & Epoch_ended(t1) @ #j
377     & LeakUSK(I,t2) @ #k
378     & Is_Epoch(t2) @ #o
379     & #j < #o
380     ==> not(Ex #l. K(m)@ #l)
381     | (Ex #l. LeakUSK(I, t1)@ #l)
382     | (Ex #l. LeakMSK(t1)@ #l)
383     | (Ex #l. LeakUpdVal(I,t1)@ #l)
384     //%other trivial leak
385     "
386
387 // Post-Compromise Security
388 //-------------------------
389
390 // Leaking usk of the target epoch yields a trivial attack.
391 lemma post_compromise_security_uskleak_attack:
392     "All m I t1 t2 #i #j #k.
393     LeakUSK(I,t1) @ #i
394     & Epoch_ended(t1) @ #j
395     & Sent(m,I) @ #k
396     & Is_Epoch(t2) @ #k
397     & #j < #k
398     ==> not(Ex #l. K(m) @ #l)
399     //| (Ex #l. LeakUSK(I, t2)@ #l)
400     | (Ex #l. LeakMSK(t2)@ #l)
401     | (Ex #l. LeakUpdVal(I,t2)@ #l)
402     %other trivial leak
403     "
404
```

```
405 // Leaking msk of the target epoch yields a trivial attack.
406 lemma post_compromise_security_mskleak_attack:
407         "All m I t1 t2 #i #j #k.
408         LeakUSK(I,t1) @ #i
409         & Epoch_ended(t1) @ #j
410         & Sent(m,I) @ #k
411         & Is_Epoch(t2) @ #k
412         & #j < #k
413         ==> not(Ex #l. K(m) @ #l)
414         | (Ex #l. LeakUSK(I, t2)@ #l)
415         //| (Ex #l. LeakMSK(t2)@ #l)
416         | (Ex #l. LeakUpdVal(I,t2)@ #l)
417         %other trivial leak
418         "
419
420 // Leaking the update value for the target epoch yields a trivial
       attack.
421 lemma post_compromise_security_tokenleak_attack:
422         "All m I t1 t2 #i #j #k.
423         LeakUSK(I,t1) @ #i
424         & Epoch_ended(t1) @ #j
425         & Sent(m,I) @ #k
426         & Is_Epoch(t2) @ #k
427         & #j < #k
428         ==> not(Ex #l. K(m) @ #l)
429         | (Ex #l. LeakUSK(I, t2)@ #l)
430         | (Ex #l. LeakMSK(t2)@ #l)
431         //| (Ex #l. LeakUpdVal(I,t2)@ #l)
432         %other trivial leak
433         "
434
435 // There is another trivial attack.
436 lemma post_compromise_security_suskleak_attack:
437     "All m I t1 t2 #i #j #k.
438     LeakUSK(I,t1) @ #i
439     & Epoch_ended(t1) @ #j
440     & Sent(m,I) @ #k
441     & Is_Epoch(t2) @ #k
442     & #j < #k
443     ==> not(Ex #l. K(m) @ #l)
444     | (Ex #l. LeakUSK(I, t2)@ #l)
445     | (Ex #l. LeakMSK(t2)@ #l)
446     | (Ex #l. LeakUpdVal(I,t2)@ #l)
447     //%other trivial leak
448     "
449
```

```
// DKER forward
//-------------

// Leaking usk of the target epoch yields a trivial attack.
lemma decryption_key_exposure_resistance_forward_uskleak_attack:
    "All I t1 t2 usk1 usk2 #i #j #k.
    Exists_Key(I, usk1, t1)@ #i
    & Is_Epoch(t1)@ #i
    & Exists_Key(I, usk2, t2)@#j
    & #i < #j
    & LeakUSK(I, t2)@ #k
    ==> not (Ex #l. K(usk1)@ #l)
    //| (Ex #l. LeakUSK(I, t1)@ #l)
    | (Ex #l. LeakMSK(t1)@ #l)
    | (Ex #l. LeakUpdVal(I,t1)@ #l)
    %other trivial leak
    "

// Leaking msk of the target epoch yields a trivial attack.
lemma decryption_key_exposure_resistance_forward_mskleak_attack:
    "All I t1 t2 usk1 usk2 #i #j #k.
    Exists_Key(I, usk1, t1)@ #i
    & Is_Epoch(t1)@ #i
    & Exists_Key(I, usk2, t2)@#j
    & #i < #j
    & LeakUSK(I, t2)@ #k
    ==> not (Ex #l. K(usk1)@ #l)
    | (Ex #l. LeakUSK(I, t1)@ #l)
    //| (Ex #l. LeakMSK(t1)@ #l)
    | (Ex #l. LeakUpdVal(I,t1)@ #l)
    %other trivial leak
    "

// Leaking the update value for the target epoch yields a trivial
    attack.
lemma decryption_key_exposure_resistance_forward_tokenleak_attack:
        "All I t1 t2 usk1 usk2 #i #j #k.
        Exists_Key(I, usk1, t1)@ #i
        & Is_Epoch(t1)@ #i
        & Exists_Key(I, usk2, t2)@#j
        & #i < #j
        & LeakUSK(I, t2)@ #k
        ==> not (Ex #l. K(usk1)@ #l)
        | (Ex #l. LeakUSK(I, t1)@ #l)
        | (Ex #l. LeakMSK(t1)@ #l)
        //| (Ex #l. LeakUpdVal(I,t1)@ #l)
        %other trivial leak
        "

// There is another trivial attack.
lemma decryption_key_exposure_resistance_forward_suskleak_attack:
        "All I t1 t2 usk1 usk2 #i #j #k.
        Exists_Key(I, usk1, t1)@ #i
```

```
502         & Is_Epoch(t1)@ #i
503         & Exists_Key(I, usk2, t2)@#j
504         & #i < #j
505         & LeakUSK(I, t2)@ #k
506         ==> not (Ex #l. K(usk1)@ #l)
507         | (Ex #l. LeakUSK(I, t1)@ #l)
508         | (Ex #l. LeakMSK(t1)@ #l)
509         | (Ex #l. LeakUpdVal(I,t1)@ #l)
510         //%other trivial leak
511         "
512
513
514 // DKER backward
515 //--------------
516
517 // Leaking usk of the target epoch yields a trivial attack.
518 lemma decryption_key_exposure_resistance_backward_uskleak_attack:
519         "All I t1 t2 usk1 usk2 #i #j #k.
520         Exists_Key(I, usk1, t1)@ #i
521         & Is_Epoch(t1)@ #i
522         & Exists_Key(I, usk2, t2)@#j
523         & #i < #j
524         & LeakUSK(I, t1)@ #k
525         ==> not (Ex #l. K(usk2)@ #l)
526         //| (Ex #l. LeakUSK(I, t2)@ #l)
527         | (Ex #l. LeakMSK(t2)@ #l)
528         | (Ex #l. LeakUpdVal(I,t2)@ #l)
529         %other trivial leak
530         "
531
532 // Leaking msk of the target epoch yields a trivial attack.
533 lemma decryption_key_exposure_resistance_backward_mskleak_attack:
534         "All I t1 t2 usk1 usk2 #i #j #k.
535         Exists_Key(I, usk1, t1)@ #i
536         & Is_Epoch(t1)@ #i
537         & Exists_Key(I, usk2, t2)@#j
538         & #i < #j
539         & LeakUSK(I, t1)@ #k
540         ==> not (Ex #l. K(usk2)@ #l)
541         | (Ex #l. LeakUSK(I, t2)@ #l)
542         //| (Ex #l. LeakMSK(t2)@ #l)
543         | (Ex #l. LeakUpdVal(I,t2)@ #l)
544         %other trivial leak
545         "
546
547 // Leaking the update value for the target epoch yields a trivial
    attack.
548 lemma
    decryption_key_exposure_resistance_backward_tokenleak_attack:
549         "All I t1 t2 usk1 usk2 #i #j #k.
550         Exists_Key(I, usk1, t1)@ #i
551         & Is_Epoch(t1)@ #i
552         & Exists_Key(I, usk2, t2)@#j
```

```
553         & #i < #j
554         & LeakUSK(I, t1)@ #k
555         ==> not (Ex #l. K(usk2)@ #l)
556         | (Ex #l. LeakUSK(I, t2)@ #l)
557         | (Ex #l. LeakMSK(t2)@ #l)
558         //| (Ex #l. LeakUpdVal(I,t2)@ #l)
559         %other trivial leak
560         "
561
562 // There is another trivial attack.
563 lemma decryption_key_exposure_resistance_backward_suskleak_attack:
564         "All I t1 t2 usk1 usk2 #i #j #k.
565         Exists_Key(I, usk1, t1)@ #i
566         & Is_Epoch(t1)@ #i
567         & Exists_Key(I, usk2, t2)@#j
568         & #i < #j
569         & LeakUSK(I, t1)@ #k
570         ==> not (Ex #l. K(usk2)@ #l)
571         | (Ex #l. LeakUSK(I, t2)@ #l)
572         | (Ex #l. LeakMSK(t2)@ #l)
573         | (Ex #l. LeakUpdVal(I,t2)@ #l)
574         //%other trivial leak
575         "
576
577 // Collusion Resistance
578 //----------------------
579
580 // Leaking msk of the target epoch yields a trivial attack.
581 lemma collusion_resistance_mskleak_attack:
582         "All I m t #i.
583         Sent(m, I)@ #i
584         & Is_Epoch(t)@ #i
585         & not (Ex #j. Create_Update(I, t)@ #j)
586         & not (Ex #j. GenUSK(I)@ #j & Is_Epoch(t)@ #j)
587         ==> not(Ex #l. K(m)@ #l)
588         //| (Ex #l. LeakMSK(t)@ #l)
589         "
590
591 end
```

B. Code for key renewal model

The following code portions detail the substitutions for the placeholders in the blueprint necessary to yield the key renewal model. Each replacement is shown with context. Placeholders with no counterpart in this overview need to be deleted without replacement in the final code.

```
1  theory iberevocbyrenewal
2  begin
3
4  builtins: asymmetric-encryption, hashing
5  functions: ibenc/4, ibdec/2
6  equations: ibdec(ibenc(m, I, pk(msk), t), h(I, msk, t)) = m
```

⋮

```
24 rule generate_user_key:
25     let
26     usk = h($I, ~msk, ~t)
27     in
28     ...
```

⋮

```
43 rule update_msk:
44     let
45     msk_new = msk_old
46     mpk_new = pk(msk_new)
47     in
48     ...
```

⋮

```
59 rule distribute_token:
60     let
61     update_value = h($I, ~msk, ~t_new)
62     in
63     [ !Epoch(~t_new)
64     , !MSK(~msk, ~t_new)
65     ]
66     --[ Is_Epoch(~t_new), Create_Update($I, ~t_new) ]->
67     [ !Update_Value($I, ~t_new, update_value) ]
```

⋮

```
70 rule update_usk:
71     let
72     usk_new = upd_val
73     in
74     ...
```

```
⋮

82 rule encrypt_message:
83     let
84     cipher = ibenc(~m, $I, mpk, ~t)
85     in
86     ...

⋮

95 rule decrypt_message:
96     let
97     usk = h($I, ~msk, ~t)
98     cipher = ibenc(m, $I, pk(~msk), ~t)
99     m_dec = ibdec(cipher, usk)
100    in
101    ...

⋮

158 // Sources lemmas
159 lemma msk_mpk_never_change [sources]:
160     "All msk mpk #i.
161     Is_MPK_MSK(msk, mpk)@ #i ==> (Ex #j. Origin_MPK_MSK(msk,
           mpk)@ #j)"
```

C. Code for individual update model (separate)

The following code portions detail the substitutions for the placeholders in the blueprint necessary to yield the individual update model with separated identity- and epoch key. Each replacement is shown with context. Placeholders with no counterpart in this overview need to be deleted without replacement in the final code.

```
1  theory IbeRevocByIndividualTokenSeparate
2  begin
3
4  builtins: asymmetric-encryption, hashing
5  functions: ibenc/4, ibdec/2, token/3
6  equations: ibdec(ibenc(m, I, pk(msk), t), <h(I, msk), token(I,
       msk, t)>) = m
```

⋮

```
25 rule generate_user_key:
26     let
27     susk = h($I, ~msk)
28     token = token($I, ~msk,~t)
29     usk = < susk, token >
30     in
31     ...
```

⋮

```
49 rule distribute_token:
50     let
51     update_value = token($I, ~msk, ~t_new)
52     in
53     [ !Epoch(~t_new)
54     , !MSK(~msk)
55     ]
56     --[ Is_Epoch(~t_new), Create_Update($I, ~t_new) ]->
57     [ !Update_Value($I, ~t_new, update_value)
58     , Out(update_value) ]
59
60 rule update_usk:
61     let
62     susk = h($I, ~msk)
63     token_old = token($I, ~msk, ~t_old)
64     usk_old = < susk, token_old >
65     usk_new = < susk, upd_val >
66     in
67     ...
```

```
⋮

rule encrypt_message:
    let
    cipher = ibenc(~m, $I, mpk, ~t)
    in
    ...

⋮

rule decrypt_message:
    let
    susk = h($I, ~msk)
    token = token($I, ~msk, ~t)
    usk = < susk, token >
    cipher = ibenc(m, $I, pk(~msk), ~t)
    m_dec = ibdec(cipher, usk)
    in
    ...

⋮

// Key  Leaks as defined in the adversary model

⋮

rule leak_static_usk:
    [ !SUSK(key, $I) ]
    --[LeakSUSK($I)]->
    [ Out(key) ]

⋮

lemma can_receive: exists-trace
    /* There exists a session in which a user can decrypt a
       message. */
    "Ex m #j. Receive(m)@ #j
    & not (Ex #i t N. LeakUSK(N,t)@ #i)
    & not (Ex #i. LeakMSK()@ #i)
    & not (Ex I t #l. LeakUpdVal(I,t)@ #l)
    & not (Ex #i N. LeakSUSK(N)@ #i)
    "
```

D. Code for individual update model (re-randomized)

The following code portions detail the substitutions for the placeholders in the blueprint necessary to yield the individual update model with re-randomization. Each replacement is shown with context. Placeholders with no counterpart in this overview need to be deleted without replacement in the final code.

```
1  theory IbeRevocByIndividualTokenRerandomized
2  begin
3
4  builtins: asymmetric-encryption, hashing
5  functions: ibenc/4, ibdec/2, token/3
6  equations: ibdec(ibenc(m, I, pk(msk), t), h(h(I, msk), token(I,
       msk, t)))= m
```

⋮

```
25 rule generate_user_key:
26     let
27     susk = h($I, ~msk)
28     token = token($I, ~msk, ~t)
29     usk = h(susk, token)
30     in
31     [ !Epoch(~t)
32     , !MSK(~msk)    ]
33     --[ GenUSK($I), Is_Epoch(~t), Exists_Key($I, usk, ~t) ]->
34     [ !USK(usk, $I, ~t)
35     , !SUSK(susk, $I)]
```

⋮

```
49 rule distribute_token:
50     let
51     update_value = token($I, ~msk, ~t_new)
52     in
53     [ !Epoch(~t_new)
54     , !MSK(~msk)
55     ]
56     --[ Is_Epoch(~t_new), Create_Update($I, ~t_new) ]->
57     [ !Update_Value($I, ~t_new, update_value)
58     , Out(update_value) ]
59
60 rule update_usk:
61     let
62     susk = h($I, ~msk)
63     token_old = token($I, ~msk, ~t_old)
```

```
64      usk_old = h(susk, token_old)
65      usk_new = h(susk, upd_val)
66      in
67      ...

⋮

76  rule encrypt_message:
77      let
78      cipher = ibenc(~m, $I, mpk, ~t)
79      in
80      ...

⋮

89  rule decrypt_message:
90      let
91      susk = h($I, ~msk)
92      token = token($I, ~msk, ~t)
93      usk = h(susk, token)
94      cipher = ibenc(m, $I, pk(~msk), ~t)
95      m_dec = ibdec(cipher, usk)
96      in
97      ...

⋮

106 // Key  Leaks as defined in the adversary model

⋮

122 rule leak_static_usk:
123     [ !SUSK(key, $I) ]
124     --[LeakSUSK($I)]->
125     [ Out(key) ]

⋮

157 lemma can_receive: exists-trace
158     /* There exists a session in which a user can decrypt a
           message. */
159     "Ex m #j. Receive(m)@ #j
160     & not (Ex #i t N. LeakUSK(N,t)@ #i)
161     & not (Ex #i. LeakMSK()@ #i)
162     & not (Ex I t #l. LeakUpdVal(I,t)@ #l)
163     & not (Ex #i N. LeakSUSK(N)@ #i)
164     "
```

E. Code for universal update model

The following code portions detail the substitutions for the placeholders in the blueprint necessary to yield the universal update model. Each replacement is shown with context. Placeholders with no counterpart in this overview need to be deleted without replacement in the final code.

```
1  theory IbeRevocByUniversalToken
2  begin
3
4  builtins: asymmetric-encryption, hashing
5  functions: ibenc/3, ibdec/2, usk_der/2, msk_der/1
6  equations: ibdec(ibenc(m, I, pk(msk_der(msk_inside))), usk_der(I,
       msk_inside)) = m
7
8  rule setup:
9      let
10     msk = msk_der(~msk_fresh)
11     mpk = pk(msk)
12     in
13     [ Fr(~msk_fresh)
14     , Fr(~t)
15     ]
16     --[ SetupOnlyOnce(), Is_Epoch(~t), Setup_MSK(msk, ~t)]->
17     [ ... ]

   ⋮

26 rule generate_user_key:
27     let
28     msk_curr = msk_der(msk_inside)
29     usk = usk_der($I, msk_inside)
30     in
31     ...

   ⋮

47 rule update_msk:
48     let
49     msk_old = msk_der(msk_old_inside)
50     msk_new = msk_der(<~upd_val, msk_old_inside>)
51     mpk_new = pk(msk_new)
52     in
53     ...

   ⋮
```

```
63 rule distribute_token:
64     let
65     msk_new = msk_der(<~upd_val, msk_old>)
66     update_value = ~upd_val
67     in
68     ...

   ⋮

75 rule update_usk:
76     let
77     usk_old = usk_der($I, old_value)
78     usk_new = usk_der($I,<~upd_val, old_value>)
79     in
80     ...

   ⋮

88 rule encrypt_message:
89     let
90     cipher = ibenc(~m, $I, mpk)
91     in
92     ...

   ⋮

101 rule decrypt_message:
102     let
103     msk = msk_der(msk_inside)
104     usk = usk_der($I, msk_inside)
105     cipher = ibenc(~m, $I, pk(msk))
106     m_dec = ibdec(cipher, usk)
107     in
108     ...

   ⋮

115 // additional rules
116 rule usk_der_math:
117     [ In(usk_der($I,exponent))
118     , In(~upd_val)]
119     --[USK_der()]->
120     [ Out(usk_der($I,<~upd_val, exponent>))]
121
122 rule usk_diff_math:
123     [In(usk_der($I,<~exponent1,exponent2>))
124     ,In(~exponent1)]
125     --[ USK_diff() ]->
126     [Out(usk_der($I,exponent2))]
127
128 rule msk_yield_usk:
129     [In(msk_der(msk_inside))]
130     -->
131     [Out(usk_der($I, msk_inside))]
```

```
⋮
//additional restrictions

restriction only_four_epochs:
    "All msk2 msk3 msk4 msk5 t2 t3 t4 t5 #i #j #k #l.
    Update_MSK(msk2, t2)@ #i
    & Update_MSK(msk3, t3)@ #j
    & Update_MSK(msk4, t4)@ #k
    & Update_MSK(msk5, t5)@ #l
    & not (msk2 = msk3)
    & not (msk3 = msk4)
    & not (msk2 = msk4)
    ==> #i = #l | #j = #l | #k = #l"

restriction USK_der_rule_only_four_times:
    "All #i #j #k #l #m.
    USK_der()@ #i
    & USK_der()@ #j
    & USK_der()@ #k
    & USK_der()@ #l
    & USK_der()@ #m
    ==> #i = #m | #j = #m | #k = #m | #l = #m"

restriction USK_diff_rule_only_four_times:
    "All #i #j #k #l #m.
    USK_diff()@ #i
    & USK_diff()@ #j
    & USK_diff()@ #k
    & USK_diff()@ #l
    & USK_diff()@ #m
    ==> #i = #m | #j = #m | #k = #m | #l = #m"

//sources lemma
lemma master_upd_only_three_times [sources]:
    "All upd_val_i msk t #j. Update_MSK(<upd_val_i, msk>, t)@ #j
    ==> ((Ex t2 #i. Setup_MSK(msk, t2)@ #i & i < j)  // @ #j is
        first update
    | (Ex  upd_val_2 msk1 t2 #i. msk = <upd_val_2, msk1> &
        Setup_MSK(msk1, t2)@ #i & i <j) // @ #j is second update
    | (Ex upd_val_3 upd_val_2 msk1 t2 #i. msk = <upd_val_3,
        upd_val_2, msk1> & Setup_MSK(msk1,t2)@ #i & i<j))" //@ #j is
        third update

⋮
lemma decryption_key_exposure_resistance_backward:
    "All I t1 t2 usk1 usk2 #i #j #k.
    Exists_Key(I, usk1, t1)@ #i
    & Is_Epoch(t1)@ #i
    & Exists_Key(I, usk2, t2)@#j
    & #i < #j
    & LeakUSK(I, t1)@ #k
    ==> not (Ex #l. K(usk2)@ #l)
```

```
340         | (Ex #l. LeakUSK(I,t2)@ #l)
341         | (Ex #l. LeakMSK(t2)@ #l)
342         | (Ex #l #y R t. LeakMSK(t)@ #l & LeakUpdVal(R,t2)@ #y)
343         "
```

⋮

```
497 lemma collusion_resistance_proof:
498     "All I m t #i.
499     Sent(m, I)@ #i
500     & Is_Epoch(t)@ #i
501     & not (Ex #j. Create_Update(I, t)@ #j)
502     & not (Ex #j. GenUSK(I)@ #j & Is_Epoch(t)@ #j)
503     ==> not(Ex #l. K(m)@ #l)
504         | (Ex #l t. LeakMSK(t)@ #l)
505         | (Ex R t2 #l #x. Consecutive(t,t2)@ #x & LeakUpdVal(R, t2)@
                #l)
506         | (Ex R #l. LeakUpdVal(R, t)@ #l)
507         "
508
509 lemma collusion_resistance_proof_token_old:
510     "All I m t #i.
511     Sent(m, I)@ #i
512     & Is_Epoch(t)@ #i
513     & not (Ex #j. Create_Update(I, t)@ #j)
514     & not (Ex #j. GenUSK(I)@ #j & Is_Epoch(t)@ #j)
515     ==> not(Ex #l. K(m)@ #l)
516         | (Ex #l t. LeakMSK(t)@ #l)
517         // | (Ex R t2 #l #x. Consecutive(t,t2)@ #x & LeakUpdVal(R,
                t2)@ #l)
518         | (Ex R #l. LeakUpdVal(R, t)@ #l)
519         "
520
521 lemma collusion_resistance_proof_token_curr:
522     "All I m t #i.
523     Sent(m, I)@ #i
524     & Is_Epoch(t)@ #i
525     & not (Ex #j. Create_Update(I, t)@ #j)
526     & not (Ex #j. GenUSK(I)@ #j & Is_Epoch(t)@ #j)
527     ==> not(Ex #l. K(m)@ #l)
528         | (Ex #l t. LeakMSK(t)@ #l)
529         | (Ex R t2 #l #x. Consecutive(t,t2)@ #x & LeakUpdVal(R, t2)@
                #l)
530         // | (Ex R #l. LeakUpdVal(R, t)@ #l)
531         "
```

List of Figures

Bibliography

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. "Efficient Lattice (H)IBE in the Standard Model". In: *Advances in Cryptology – EUROCRYPT 2010*. Ed. by Henri Gilbert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 553–572. ISBN: 978-3-642-13190-5. DOI: 10.1007/978-3-642-13190-5_28.

[AB22] Fatemeh Alidadi Shamsabadi and Shaghayegh Bakhtiari Chehelcheshmeh. "A cloud-based mobile payment system using identity-based signature providing key revocation". In: *The Journal of Supercomputing* 78.2 (2022), pp. 2503–2527. ISSN: 1573-0484. DOI: 10.1007/s11227-021-03830-4. URL: https://link.springer.com/article/10.1007/s11227-021-03830-4.

[Ana+20] Hiroaki Anada et al. "Key-updatable public-key encryption with keyword search (Or: How to realize PEKS with efficient key updates for IoT environments)". In: *International Journal of Information Security* 19.1 (2020), pp. 15–38. ISSN: 1615-5270. DOI: 10.1007/s10207-019-00441-2.

[Bae+04] Joonsang Baek et al. "A survey of identity-based cryptography". In: *Proc. of Australian Unix Users Group Annual Conference*. 2004, pp. 95–102. ISBN: 0-9577532-6-8. URL: https://books.google.de/books?id=Lr0XnWmD0zgC&lpg=PA95&ots=9HhZbX_18R&dq=A%20Survey%20of%20Identity-Based%20Cryptography&lr&pg=PP4#v=onepage&q&f=false.

[Bar+21] Manuel Barbosa et al. "SoK: Computer-Aided Cryptography". In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021, pp. 777–795. DOI: 10.1109/SP40001.2021.00008.

[Bar20] Elaine Barker. "Recommendation for key management". In: *National Institute of Standards and Technology* (2020). DOI: 10.6028/NIST.SP.800-57pt1r5. URL: https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final.

[Bas+14] David Basin et al. "ARPKI: Attack Resilient Public-Key Infrastructure". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2014. DOI: 10.1145/2660267.2660298.

[Bas+17] David Basin et al. "Symbolically analyzing security protocols using tamarin". In: *ACM SIGLOG News* 4.4 (Nov. 2017), pp. 19–30. DOI: 10.1145/3157831.3157835. URL: https://doi.org/10.1145/3157831.3157835.

[BHS19] David A. Basin, Lucca Hirschi, and Ralf Sasse. "Symbolic Analysis of Identity-Based Protocols Dedicated to Catherine A. Meadows". In: *Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows*. Ed. by Joshua D. Guttman et al. Lecture Notes in Computer Science. Springer, 2019, pp. 112–134. ISBN: 978-3-030-19051-4. DOI: 10.1007/978-3-030-19052-1_9.

[Bla16] Bruno Blanchet. "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif". In: *Foundations and Trends® in Privacy and Security* 1.1-2 (2016), pp. 1–135. ISSN: 2474-1558. DOI: 10.1561/3300000004. URL: http://dx.doi.org/10.1561/3300000004.

[BGK08] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. "Identity-based encryption with efficient revocation". In: *Proceedings of the 15th ACM Conference on Computer and Communications Security*. Ed. by Peng Ning, Paul Syverson, and Somesh Jha. New York, NY: ACM, 2008, pp. 417–426. ISBN: 9781595938107. DOI: 10.1145/1455770.1455823.

[BF01] Dan Boneh and Matt Franklin. "Identity-Based Encryption from the Weil Pairing". In: *Advances in cryptology*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Berlin: Springer, 2001, pp. 213–229. ISBN: 978-3-540-42456-7. DOI: 10.1007/3-540-44647-8_13.

[BG21] Colin Boyd and Kai Gellert. "A Modern View on Forward Security". In: *The Computer Journal* 64.4 (2021), pp. 639–652. ISSN: 1460-2067. DOI: 10.1093/comjnl/bxaa104.

[BDS17] Alessandro Bruni, Eva Drewsen, and Carsten Schürmann. "Towards a Mechanized Proof of Selene Receipt-Freeness and Vote-Privacy". In: *Electronic Voting*. Ed. by Robert Krimmer et al. Cham: Springer International Publishing, 2017, pp. 110–126. ISBN: 978-3-319-68687-5. DOI: 10.1007/978-3-319-68687-5_7.

[CCA23] Cas Cremers, Charlie Jacomme, and Aurora Naska. "Formal Analysis of Session-Handling in Secure Messaging: Lifting Security from Sessions to Conversations". In: *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, 2023, pp. 1235–1252. ISBN: 978-1-939133-37-3. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/cremers-session-handling.

[CD23] Wouter Castryck and Thomas Decru. "An Efficient Key Recovery Attack on SIDH". In: *Advances in Cryptology – EUROCRYPT 2023*. Ed. by Carmit Hazay and Martijn Stam. Cham: Springer Nature Switzerland, 2023, pp. 423–447. ISBN: 978-3-031-30589-4. DOI: 10.1007/978-3-031-30589-4_15.

[Cha+16] Rohit Chadha et al. "Automated verification of equivalence properties of cryptographic protocols". In: *ACM Transactions on Computational Logic (TOCL)* 17.4 (2016), pp. 1–32. DOI: 10.1145/2926715.

[Che+12] Jie Chen et al. "Revocable Identity-Based Encryption from Lattices". In: *Information security and privacy*. Ed. by Willy Susilo, Yi Mu, and Jennifer Seberry. Vol. 7372. Lecture Notes in Computer Science. Berlin: Springer, 2012, pp. 390–403. ISBN: 978-3-642-31447-6. DOI: 10.1007/978-3-642-31448-3_29.

[CZ15] Shantian Cheng and Juanyang Zhang. "Adaptive-ID Secure Revocable Identity-Based Encryption from Lattices via Subset Difference Method". In: *Information security practice and experience*. Ed. by Javier Lopez and Yongdong Wu. Vol. 9065. Lecture Notes in Computer Science. Cham: Springer, 2015, pp. 283–297. ISBN: 978-3-319-17532-4. DOI: 10.1007/978-3-319-17533-1_20.

[CKR18] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. "DEEPSEC: deciding equivalence properties in security protocols theory and practice". In: *2018 IEEE symposium on security and privacy (SP)*. IEEE. 2018, pp. 529–546. DOI: 10.1109/SP.2018.00033.

[Cho+19] Joo Cho et al. "Towards Quantum-resistant Virtual Private Networks". In: *crypto day matters 31*. Ed. by Marcel Selhorst, Daniel Loebenberger, and Michael Nüsken. Bonn: Gesellschaft für Informatik e.V. / FG KRYPTO, 2019. DOI: 10.18420/cdm-2019-31-22.

[CCG16] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. "On Post-compromise Security". In: *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. IEEE, 2016. DOI: 10.1109/csf.2016.19.

[Coh+20] Katriel Cohn-Gordon et al. "A Formal Security Analysis of the Signal Messaging Protocol". In: *Journal of Cryptology* 33.4 (2020), pp. 1914–1983. ISSN: 1432-1378. DOI: 10.1007/s00145-020-09360-1. URL: https://link.springer.com/article/10.1007/s00145-020-09360-1.

[CHK21] Cas Cremers, Britta Hale, and Konrad Kohbrok. "The Complexities of Healing in Secure Group Messaging: Why Cross-Group Effects Matter". In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1847–1864. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/system/files/sec21-cremers.pdf.

[CM12] Cas Cremers and Sjouke Mauw. *Operational semantics and verification of security protocols*. Information security and cryptography. Berlin and Heidelberg: Springer, 2012. ISBN: 978-3-540-78635-1. DOI: 10.1007/978-3-540-78636-8.

[DY83] D. Dolev and A. Yao. "On the security of public key protocols". In: *IEEE Transactions on Information Theory* 29.2 (1983), pp. 198–208. ISSN: 0018-9448. DOI: 10.1109/TIT.1983.1056650.

[Eat+22] Edward Eaton et al. "Towards Post-Quantum Key-Updatable Public-Key Encryption via Supersingular Isogenies". In: *Selected Areas in Cryptography*. Ed. by Riham AlTawy and Andreas Hülsing. Cham: Springer International Publishing, 2022, pp. 461–482. ISBN: 978-3-030-99277-4. DOI: 10.1007/978-3-030-99277-4_22.

[Eur22] European Telecommunications Standards Institute. *Guide to Identity-Based Cryptography*. 2022. URL: http://www.etsi.org-/deliver/etsi_tr/103700_103799/103719/ (visited on 05/13/2024).

[GG09] David Galindo and Flavio D. Garcia. "A Schnorr-Like Lightweight Identity-Based Signature Scheme". In: *Progress in Cryptology – AFRICACRYPT 2009*. Ed. by Bart Preneel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 135–148. ISBN: 978-3-642-02384-2. DOI: 10.1007/978-3-642-02384-2_9.

[Gaz+21a] Stefan-Lukas Gazdag et al. "A formal analysis of IKEv2's post-quantum extension". In: *Annual Computer Security Applications Conference*. ACM Digital Library. New York, NY, United States: Association for Computing Machinery, 2021, pp. 91–105. ISBN: 9781450385794. DOI: 10.1145/3485832.3485885.

[Gaz+21b] Stefan-Lukas Gazdag et al. “A formal analysis of IKEv2’s post-quantum extension”. In: *Annual Computer Security Applications Conference*. ACM Digital Library. New York,NY,United States: Association for Computing Machinery, 2021, pp. 91–105. ISBN: 9781450385794. DOI: 10.1145/3485832.3485885.

[Gaz+21c] Stefan-Lukas Gazdag et al. “Entangled Secrets”. In: *Linux Magazine* 247 (2021), pp. 16–19. URL: https://www.linux-magazine.com/Issues/2021/247/Quantum-Computing-and-Encryption (visited on 07/02/2021).

[Gaz+21d] Stefan-Lukas Gazdag et al. “Migration zu quantenresistenter IT”. In: *Sicherheit in vernetzten Systemen: 28. DFN-Konferenz*. Ed. by Albrecht Ude. Books on Demand, 2021, E-1–E-23. ISBN: 9783753448848.

[Gaz+21e] Stefan-Lukas Gazdag et al. “Migration zu quantenresistenter IT”. In: *Linux Magazin* 04 (2021), pp. 84–88. URL: https://www.linux-magazin.de/ausgaben/2021/04/kryptographie/ (visited on 07/02/2021).

[Gaz+21f] Stefan-Lukas Gazdag et al. “Quantensichere IT: ein Blick in die Glaskugel”. In: *DFN Mitteilungen* 99 (2021), pp. 34–38. URL: https://www.dfn.de/fileadmin/5Presse/DFNMitteilungen/DFN_Mitteilungen_99.pdf (visited on 07/02/2021).

[Gaz+23] Stefan-Lukas Gazdag et al. “Quantum-Resistant MACsec and IPsec for Virtual Private Networks”. In: *Security Standardisation Research*. Ed. by Felix Günther and Julia Hesse. Cham: Springer Nature Switzerland, 2023, pp. 1–21. ISBN: 978-3-031-30731-7. DOI: 10.1007/978-3-031-30731-7_1.

[gGG18] Nils gentschen Felde, Sophia Grundner-Culemann, and Tobias Guggemos. “Authentication in dynamic groups using identity-based signatures”. In: *14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. Limassol, Cyprus, 2018. DOI: 10.1109/WiMOB.2018.8589148.

[Gru22] Sophia Grundner-Culemann. “A Survey of Revocation Mechanisms in Identity-based Cryptography”. In: *crypto day matters 34*. Ed. by Daniel Loebenberger and Michael Nüsken. Bonn: Gesellschaft für Informatik e.V. / FG KRYPTO, 2022. DOI: 10.18420/cdm-2022-34-03.

[GK21] Sophia Grundner-Culemann and Dieter Kranzlmüller. “EUF-ID-UPD-CMA: A security notion for key-updatable identity-based signature schemes”. In: *crypto day matters 32*. Ed. by Stefan-Lukas Gazdag, Daniel Loebenberger, and Michael Nüsken. Bonn: Gesellschaft für Informatik e.V. / FG KRYPTO, 2021. DOI: 10.18420/cdm-2021-32-34.

[Gug20] Tobias Guggemos. *Efficient signature verification and key revocation using identity based cryptography*. Ludwig-Maximilians-Universität München, 2020. DOI: 10.5282/edoc.27261. URL: https://edoc.ub.uni-muenchen.de/27261/.

[GG21] Tobias Guggemos and Sophia Grundner-Culemann. “group Identity Based Signatures: Efficiently revoking signing keys in communication groups”. In: *crypto day matters 32*. Ed. by Stefan-Lukas Gazdag, Daniel Loebenberger, and Michael Nüsken. Bonn: Gesellschaft für Informatik e.V. / FG KRYPTO, 2021. DOI: 10.18420/cdm-2021-32-33.

[Han+05] Yumiko Hanaoka et al. "Identity-Based Hierarchical Strongly Key-Insulated Encryption and Its Application". In: *Advances in cryptology - ASIACRYPT 2005*. Ed. by Bimal Roy. Vol. 3788. Lecture Notes in Computer Science. Berlin: Springer, 2005, pp. 495–514. ISBN: 978-3-540-30684-9. DOI: 10.1007/11593447_27.

[Hes02] Florian Hess. *Exponent Group Signature Schemes and Efficient Identity Based Signature Schemes Based on Pairings*. 2002. URL: https://eprint.iacr.org/2002/012.

[Hes03] Florian Hess. *Efficient Identity Based Signature Schemes Based on Pairings*. Ed. by Kaisa Nyberg and Howard Heys. Berlin, Heidelberg, 2003. DOI: 10.1007/3-540-36492-7_20.

[ISO18] ISO Central Secretary. *ISO/IEC 27000:2018: Information technology - Security techniques - Information security management systems - Overview and vocabulary: Standard*. Geneva, CH, Feb. 2018. URL: https://www.iso.org/standard/73906.html (visited on 12/29/2021).

[KNT20] Nadim Kobeissi, Georgio Nicolas, and Mukesh Tiwari. "Verifpal: Cryptographic protocol analysis for the real world". In: *Progress in Cryptology–INDOCRYPT 2020: 21st International Conference on Cryptology in India, Bangalore, India, December 13–16, 2020, Proceedings 21*. Springer. 2020, pp. 151–202. DOI: 10.1007/978-3-030-65277-7_8.

[LT18] Anja Lehmann and Björn Tackmann. "Updatable Encryption with Post-Compromise Security". In: *Advances in Cryptology - EUROCRYPT 2018*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 685–716. ISBN: 978-3-319-78371-0. DOI: 10.1007/978-3-319-78372-7_22.

[LQ03] Benoît Libert and Jean-Jacques Quisquater. "Efficient revocation and threshold pairing based cryptosystems". In: *Proceedings of the twenty-second annual symposium on Principles of distributed computing*. Ed. by Elizabeth Borowsky. New York, NY: ACM, 2003. ISBN: 1581137087. DOI: 10.1145/872035.872059.

[LQ04] Benoît Libert and Jean-Jacques Quisquater. "What Is Possible with Identity Based Cryptography for PKIs and What Still Must Be Improved". In: Springer, Berlin, Heidelberg, 2004, pp. 57–70. DOI: 10.1007/978-3-540-25980-0_5. URL: https://link.springer.com/chapter/10.1007/978-3-540-25980-0_5.

[LV09] Benoît Libert and Damien Vergnaud. "Adaptive-ID Secure Revocable Identity-Based Encryption". In: *Topics in cryptology - CT-RSA 2009*. Ed. by Marc Fischlin. Vol. 5473. Lecture Notes in Computer Science. Berlin: Springer, 2009, pp. 1–15. ISBN: 978-3-642-00861-0. DOI: 10.1007/978-3-642-00862-7_1.

[Liu+16] Zhenhua Liu et al. "Revocable and strongly unforgeable identity-based signature scheme in the standard model". In: *Security and Communication Networks* 9.14 (2016), pp. 2422–2433. ISSN: 19390114. DOI: 10.1002/sec.1513.

[Low96] Gavin Lowe. "Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR". In: *Tools and algorithms for the construction and analysis of systems*. Ed. by Bernhard Steffen and Tiziana Margaria. Vol. 1055. Lecture Notes in Computer Science. Berlin: Springer, 1996, pp. 147–166. ISBN: 978-3-540-61042-7. DOI: 10.1007/3-540-61042-1_43.

[ML20] Xuecheng Ma and Dongdai Lin. "Generic Constructions of Revocable Identity-Based Encryption". In: *Information Security and Cryptology*. Ed. by Zhe Liu and Moti Yung. Cham: Springer International Publishing, 2020, pp. 381–396. ISBN: 978-3-030-42921-8. DOI: 10.1007/978-3-030-42921-8_22.

[MG22] Nils Mäurer and Sophia Grundner-Culemann. "Formal Verification of the LDACS MAKE Protocol". In: *crypto day matters 34*. Ed. by Daniel Loebenberger and Michael Nüsken. Bonn: Gesellschaft für Informatik e.V. / FG KRYPTO, 2022. DOI: 10.18420/cdm-2022-34-24.

[Mäu+22] Nils Mäurer et al. "Security in Digital Aeronautical Communications A Comprehensive Gap Analysis". In: *International Journal of Critical Infrastructure Protection* 38 (2022), p. 100549. ISSN: 1874-5482. DOI: 10.1016/j.ijcip.2022.100549. URL: https://www.sciencedirect.com/science/article/pii/S187454822200035X.

[Mea11] Catherine Meadows. "Formal Analysis of Cryptographic Protocols". In: *Encyclopedia of Cryptography and Security*. Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Boston, MA: Springer US, 2011, pp. 490–492. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_876. URL: https://doi.org/10.1007/978-1-4419-5906-5_876.

[NS78] Roger M. Needham and Michael D. Schroeder. "Using encryption for authentication in large networks of computers". In: *Communications of the ACM* 21.12 (1978), pp. 993–999. ISSN: 0001-0782. DOI: 10.1145/359657.359659.

[WHA99] D. Wallner, E. Harder, and R. Agee. *Key Management for Multicast: Issues and Architectures*. RFC 2627 (Informational). RFC. Fremont, CA, USA: RFC Editor, June 1999. DOI: 10.17487/RFC2627. URL: https://www.rfc-editor.org/rfc/rfc2627.txt.

[Coo+08] D. Cooper et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280 (Proposed Standard). RFC. Updated by RFCs 6818, 8398, 8399. Fremont, CA, USA: RFC Editor, May 2008. DOI: 10.17487/RFC5280. URL: https://www.rfc-editor.org/rfc/rfc5280.txt.

[TC11] S. Turner and L. Chen. *Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms*. RFC 6151 (Informational). RFC. Fremont, CA, USA: RFC Editor, Mar. 2011. DOI: 10.17487/RFC6151. URL: https://www.rfc-editor.org/rfc/rfc6151.txt.

[SW05] Amit Sahai and Brent Waters. "Fuzzy Identity-Based Encryption". In: *Advances in Cryptology – EUROCRYPT 2005*. Ed. by Ronald Cramer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 457–473. ISBN: 978-3-540-32055-5. DOI: 10.1007/11426639_27.

[Sch+21] Antonius Scherer et al. “OnCall Operator Scheduling for Satellites with Grover’s Algorithm”. In: *Computational Science - ICCS 2021 : 21st International Conference, Krakow, Poland, June 16-18, 2021, Proceedings, Part VI*. Ed. by Maciej Paszynski. Vol. 12747. Lecture Notes in Computer Science. Cham: Springer, 2021, pp. 17–29. ISBN: 978-3-030-77979-5. DOI: 10.1007/978-3-030-77980-1_2.

[Sch+12] Benedikt Schmidt et al. “Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties”. In: *2012 IEEE 25th Computer Security Foundations Symposium*. 2012, pp. 78–94. DOI: 10.1109/CSF.2012.25.

[Sch98] Bruce Schneier. *Crypto-Gram: A free monthly newsletter providing summaries, analyses, insights, and commentaries on cryptography and computer security.* 1998. URL: https://www.schneier.com/crypto-gram/archives/1998/1015.html#cipherdesign (visited on 08/30/2023).

[Sch11] Bruce Schneier. *Schneier’s Law*. 2011. URL: https://www.schneier.com/blog/archives/2011/04/schneiers_law.html (visited on 08/30/2023).

[SE13a] Jae Hong Seo and Keita Emura. “A Remark on “ Efficient Revocable ID-Based Encryption with a Public Channel””. In: *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences* E96-A.11 (2013), pp. 2282–2285. ISSN: 0916-8508. DOI: 10.1587/transfun.E96.A.2282. URL: https://search.ieice.org/bin/summary.php?id=e96-a_11_2282.

[SE13b] Jae Hong Seo and Keita Emura. “Revocable Identity-Based Encryption Revisited: Security Model and Construction”. In: *Public-key cryptography - PKC 2013*. Ed. by Kaoru Kurosawa. Vol. 7778. Lecture Notes in Computer Science. Berlin: Springer, 2013, pp. 216–234. ISBN: 978-3-642-36361-0. DOI: 10.1007/978-3-642-36362-7_14.

[SE14] Jae Hong Seo and Keita Emura. “Revocable Identity-Based Encryption with Rejoin Functionality”. In: *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences* E97.A.8 (2014), pp. 1806–1809. ISSN: 0916-8508. DOI: 10.1587/transfun.E97.A.1806.

[Sha85] Adi Shamir. “Identity-Based Cryptosystems and Signature Schemes”. In: Springer, Berlin, Heidelberg, 1985, pp. 47–53. DOI: 10.1007/3-540-39568-7_5. URL: https://link.springer.com/chapter/10.1007/3-540-39568-7_5.

[Shw+11] Shweta Agrawal et al. *Fuzzy Identity Based Encryption from Lattices*. Cryptology ePrint Archive, Paper 2011/414. 2011. URL: https://eprint.iacr.org/2011/414.

[SVR21] Srijanee Mookherji, Vanga Odelu, and Rajendra Prasath. “Modelling IBE-based Key Exchange Protocol using Tamarin Prover”. In: *Cryptology ePrint Archive* (2021). URL: https://eprint.iacr.org/2021/1598.

[Sta+23] Korbinian Staudacher et al. “Reducing 2-QuBit Gate Count for ZX-Calculus based Quantum Circuit Optimization”. In: *Electronic Proceedings in Theoretical Computer Science* 394 (Nov. 2023), pp. 29–45. ISSN: 2075-2180. DOI: 10.4204/eptcs.394.3. URL: http://dx.doi.org/10.4204/EPTCS.394.3.

[TW17] Atsushi Takayasu and Yohei Watanabe. "Lattice-Based Revocable Identity-Based Encryption with Bounded Decryption Key Exposure Resistance". In: *Information Security and Privacy*. Ed. by Josef Pieprzyk and Suriadi Suriadi. Vol. 10342. Lecture Notes in Computer Science. Cham and s.l.: Springer International Publishing, 2017, pp. 184–204. ISBN: 978-3-319-60054-3. DOI: 10.1007/978-3-319-60055-0_10.

[TW21] Atsushi Takayasu and Yohei Watanabe. "Revocable identity-based encryption with bounded decryption key exposure resistance: Lattice-based construction and more". In: *Theoretical Computer Science* 849 (2021), pp. 64–98. ISSN: 03043975. DOI: 10.1016/j.tcs.2020.10.010.

[Tel23] Bernhard Tellenbach. "Identity-Based Cryptography". In: *Trends in Data Protection and Encryption Technologies*. Ed. by Valentin Mulder et al. Cham: Springer Nature Switzerland, 2023, pp. 59–64. ISBN: 978-3-031-33386-6. DOI: 10.1007/978-3-031-33386-6_12.

[Tri20] Tristan Ninet. "Formal verification of the Internet Key Exchange (IKEv2) security protocol". PhD thesis. Université Rennes 1, 2020. URL: https://tel.archives-ouvertes.fr/tel-02882167/.

[TTW12] Tung-Tso TSAI, Yuh-Min Tseng, and Tsu-Yang Wu. "A Fully Secure Revocable ID-Based Encryption in the Standard Model". In: *Informatica* 23.3 (2012), pp. 487–505. ISSN: 0868-4952. DOI: 10.15388/Informatica.2012.371. URL: https://content.iospress.com/articles/informatica/inf23-3-09.

[TTW13] Tung-Tso TSAI, Yuh-Min Tseng, and Tsu-Yang Wu. "Provably secure revocable ID-based signature in the standard model". In: *Security and Communication Networks* 6.10 (2013), pp. 1250–1260. ISSN: 19390114. DOI: 10.1002/sec.696.

[TT12] Y.-M. Tseng and T.-T. Tsai. "Efficient Revocable ID-Based Encryption with a Public Channel". In: *The Computer Journal* 55.4 (2012), pp. 475–486. ISSN: 0010-4620. DOI: 10.1093/comjnl/bxr098.

[VP17] Mathy Vanhoef and Frank Piessens. "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 1313–1328. ISBN: 9781450349468. DOI: 10.1145/3133956.3134027.

[Whi+17] Jorden Whitefield et al. "Formal Analysis of V2X Revocation Protocols". In: Springer, Cham, 2017, pp. 147–163. DOI: 10.1007/978-3-319-68063-7_10. URL: https://link.springer.com/chapter/10.1007/978-3-319-68063-7_10.

[XWW20] Congge Xie, Jian Weng, and Jinming Wen. "Scalable Revocable Identity-Based Signature Scheme with Signing Key Exposure Resistance from Lattices". In: *Security and Communication Networks* 2020 (2020), pp. 1–11. ISSN: 19390114. DOI: 10.1155/2020/1743421.

[YSM10] Tsz Hon Yuen, Willy Susilo, and Yi Mu. "How to construct identity-based signatures without the key escrow problem". In: *International Journal of Information Security* 9.4 (2010), pp. 297–311. ISSN: 1615-5270. DOI: 10.1007/s10207-010-0110-5. URL: https://link.springer.com/article/10.1007/s10207-010-0110-5.

[Zha+12] Yunmei Zhang et al. "Efficient Escrow-Free Identity-Based Signature". In: *Provable Security*. Ed. by Tsuyoshi Takagi et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 161–174. ISBN: 978-3-642-33272-2. DOI: 10.1007/978-3-642-33272-2_11.

Acronyms

MPK Master Public Key 1, 11, 12, 37–39, 44, 46, 51–54, 61, 78, 82

msk Master Secret Key 1, 11, 12, 37–39, 42, 44, 52, 53, 61, 62, 64, 78, 84–86, 88, 89

usk user secret key 11, 12, 37–39, 44, 48, 51, 87

CA Certificate Authority 1, 10

CRL Certificate Revocation List 12

HIBE Hierarchical Identity-based Encryption 22

IBC Identity-based Cryptography ix, xiii, xiv, 1–3, 9, 10, 12, 19, 20, 23, 25, 27, 28, 30, 32–36, 38, 39, 41, 45, 49, 50, 52, 53, 55, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 88, 91–93

IBE Identity-based Encryption 12, 20

IBS Identity-based Signatures 12

ID Identity 12

IEC International Electrotechnical Commission 9

IoT Internet of Things 1

ISO International Organization for Standardization 9

LKH Logical Key Hierarchy 31

NIST National Institute of Standards and Technology 13, 32–34

PKI Public Key Infrastructure 1, 9, 10, 12

RIBE Revocable Identity-based Encryption 21, 22

TTP Trusted Third Party ix, 1, 11–13, 21, 23, 30, 31, 35, 37, 38, 41, 42, 44–47, 51, 59, 61–63, 65, 66, 69, 72, 73, 82, 86, 91