

Bachelor's Thesis

Privacy Friendly App Usage Metrics

Semester: Spring 2026



Date: 11.06.2026

Project Team: Dorian Di Pierro
Silvan Lendi
Fabio Zahner

Project Advisor: Dr. Daniel Tschudi
Co-Advisor: Alice Glaus
Examiner: Prof. Dr. Chen-Da Liu-Zhang



School of Computer Science
OST Eastern Switzerland University of Applied Sciences

Abstract

Threema is a cross-platform encrypted messaging app developed by Threema GmbH in Switzerland. It uses end-to-end encryption by default, requires no phone number or email to register, and is fully open source. Because of this strong privacy focus, Threema collects no user data, which means the product team has no insight into how the app is actually used. This thesis designs, implements, and evaluates a privacy-preserving, opt-in telemetry system for Threema Desktop to address that gap.

Six usage metrics are collected across engagement, retention, communication, feature adoption, performance, and social graph categories. Re-identification risk is controlled through bucketing, Laplace-mechanism differential privacy noise, identifier-free transmission with padding, and a dynamic entropy budget that limits how many metrics are reported until enough users have opted in. Shannon entropy serves as a pre-collection proxy for k-anonymity, since the true value cannot be computed without an existing dataset. Random metric selection per reporting window allows more metrics to be tracked over time without exceeding the budget.

The prototype consists of three components: a client-side metrics package embedded in the Threema Desktop app, a backend server that receives and aggregates the transmitted data, and an administration tool that lets operators tune privacy parameters and inspect results. All technical requirements have been met and adequate measures have been taken to test the system performance. Mock-data tests validate the k-anonymity formulas and network measurements confirm that padding removes packet-length side channels.

Several open issues remain. The k-anonymity guarantees are estimates based on entropy calculations and cannot be verified until real opt-in data is available. Submission abuse is currently mitigated only through IP-based rate limiting; a zero-knowledge proof scheme would allow the server to authenticate genuine Threema clients without learning their identity, which is a stronger guarantee. The current metrics package only covers Threema Desktop; extending it to mobile platforms would require a redesign in Kotlin and Swift and introduces additional complexity around multi-device submission. Pending these open points, the system is a viable basis for Threema to evaluate a production integration.

Contents

Abstract	i
1 Problem Description	1
1.1 The Threema App	1
1.2 Lack of Metrical Evidence	2
1.3 Project Goal	2
2 Preliminaries	3
2.1 Privacy Preserving Concepts	3
2.1.1 K-Anonymity	3
2.1.2 Differential Privacy	4
2.2 Data Protection Regulation	5
2.3 Data Submission Abuse Prevention	6
3 Solution Design	7
3.1 Implementation Platform	7
3.2 Metrics	7
3.2.1 Goals of the Telemetry	7
3.2.2 Criteria for Evaluating Metrics	8
3.2.2.1 Setting a k-anonymity Target	9
3.2.2.2 Shannon Entropy as a Combination Budget	10
3.2.2.3 Entropy Control through Bucketing	11
3.2.2.4 Applicability of l-diversity and t-closeness	11
3.2.2.5 Summary	12
3.2.3 Metrics Selection	13
3.2.3.1 Selected Metrics	14
3.2.3.2 Summary	15
3.3 Random Sampling	15
3.3.1 Random Metric Selection	15
3.3.1.1 All users report the same metrics	15
3.3.1.2 Random subset of metrics per reporting window	16
3.3.1.3 Fixed random subset of metrics per user	16
3.3.1.4 Conclusion	16
3.3.2 K-anonymity under Random Sampling	16

3.3.3	Formulas for k-anonymity	17
3.4	Dynamic Privacy Budget	19
3.5	Opt-In	20
3.5.1	Consent Rate vs Opt-in Rate	20
3.5.2	Placement	20
3.5.2.1	Settings	21
3.5.2.2	Popup Design	21
3.5.3	Summary	22
3.6	Testing Concept	22
3.6.1	Testing Methods	22
3.6.2	Summary	23
4	System Development	24
4.1	Functional Requirements	24
4.1.1	Threema App Client Requirements	25
4.1.2	Server Requirements	27
4.1.3	Administration Tool Requirements	27
4.2	Market Study	28
4.2.1	Criteria Fulfillment	28
4.2.2	Existing Tools	29
4.3	Data Transmission	30
4.3.1	Defining Bucket Ranges	30
4.3.2	Visualization	31
4.4	Technology Decisions	32
4.4.1	Metrics Package	32
4.4.2	Metrics Server	33
4.4.3	Metrics Database	33
4.4.4	Visualization	34
4.5	Metrics Definition	34
4.5.1	Updating the Schema	34
4.6	System Architecture	35
4.7	Threema Desktop App	35
4.7.1	Package architecture	36
4.7.1.1	Collection Layer	37
4.7.1.2	Data Access Layer	37
4.7.1.3	Scheduling Layer	38
4.7.1.4	Calculation Layer	39
4.7.1.5	Privacy Layer	39
4.7.1.6	Network Layer	39
4.7.1.7	Configuration Layer	40
4.7.2	Opt-in	40
4.8	Metrics Server	40
4.8.1	Data Model	41

4.8.2	The API	41
4.8.3	Telemetry Submission Flow	42
4.8.4	Privacy Parameter Computation	42
4.8.5	Initialization and Configuration	43
4.9	Multi-device Support	43
4.9.1	Platform-specific Metric as Additional Entropy	43
4.9.2	Transmitting Additional Information	43
4.9.3	Solution	43
4.10	Administration Dashboards	44
4.11	Configuration Frontend	45
4.11.1	Structure	45
4.11.2	Metric Weight Configuration	45
4.11.3	Privacy Budget Configuration	46
4.11.4	Styling	47
4.12	Application Flow	47
5	Evaluation and Results	49
5.1	Testing Results	49
5.1.1	Mock Data Evaluation	49
5.1.2	Unit Test Results	50
5.1.3	Network Test Results	50
5.1.4	Discussion	51
5.1.5	Conclusion	51
5.2	Requirements Fulfillment	51
5.3	Privacy Evaluation	53
5.3.1	Re-identification Risk	53
5.3.2	Network-Level Information Leakage	54
5.4	Limitations	54
5.5	Discussion	54
6	Conclusion	56
6.1	Summary	56
6.2	Future Work	57
6.3	Concluding Remarks	58
	Bibliography	59
	List of Tables	64
	List of Figures	66
	Tool Usage Collection	68
	Acronyms	69

A	Additional Artifacts	70
A.1	List of Metrics	70
A.1.1	Engagement	70
A.1.2	Retention	71
A.1.3	Communication	71
A.1.4	Acquisition & Growth	71
A.1.5	Feature Adoption	72
A.1.6	Technical / Performance	72
A.1.7	Social Graph	72
A.1.8	Notification & Reachability	73
A.2	Comparison of Analytics Platforms	73
A.3	Mockups	75
A.3.1	Threema Desktop	75
A.4	Models	76
A.4.1	Entity Relationship Diagram	76
A.4.2	Threat Model	76
A.4.2.1	Assets	76
A.4.2.2	Actors	76
A.4.2.3	LIDDUN Threat Analysis	77
A.5	Simulated Testing	78
A.5.1	Basic K-anonymity Testing	78
A.5.2	K-anonymity Methods with Random Sampling	79
A.5.3	Subset Method with Large Datasets	81
A.5.4	Alternate Entropy Calculation Methods	83
A.6	Formulas and Calculations	84
A.6.1	Calculating System State	84
A.6.2	From k_{base} to k_{min}	85
A.6.3	Calculating privacy budget	85
A.6.3.1	”Simple” Mode	85
A.6.3.2	Average and Minimum K-Anonymity Modes	85
A.7	Opt-in	86
A.7.1	Popup Designs	86
A.7.2	Implemented Setting	88
A.8	Configuration Frontend Screens	88
A.9	Grafana Dashboards	92
A.10	OpenAPI Specification	97
A.11	Example Metrics Definition File	98

Chapter 1

Problem Description

This chapter gives an overview of the problem that this project aims to solve. It starts with a brief introduction to the Threema app and its commitment to user privacy, which leads to a lack of metrical evidence for understanding user behavior and improving the app.

1.1 The Threema App

Threema is a cross-platform encrypted instant messaging app developed by Threema GmbH, a Swiss company with headquarters in Pfäffikon SZ. The app is designed to provide secure communication with a special focus on user privacy. It achieves this by using end-to-end encryption, which ensures that only the sender and recipient(s) can read sent messages. This is combined with the open-sourcing of all client applications and regular audits by independent companies[Thr26b] to provide transparency and build trust with users.

The key elements of Threema are summarized in Table 1.1.

Table 1.1: Key elements of Threema.

Selling Point	Description
Privacy Focus	Threema does not require users to provide an email or phone number to create an account.
Encryption	All messages, calls, and media shared through Threema are end-to-end encrypted by default.
Open Source	The client applications are open source, allowing for independent security audits and fostering trust among users.
Data Protection	Fully compliant with data protection regulations and subject to strong Swiss privacy laws.

1.2 Lack of Metrical Evidence

Because Threema is designed with a strong focus on user privacy, it does not collect or store any user data that could be used for analytics. It is important to mention that Threema allows optional crash reports, which they use for stability improvements. While this approach provides a high level of privacy for users, it also means that Threema lacks metrical evidence to make informed decisions about areas of improvement apart from stabilizing the application.

1.3 Project Goal

The goals of this bachelor thesis are to:

1. Design and plan a concept for a privacy-friendly, opt-in app usage metrics system for Threema
2. Implement a prototype of such a system on at least one platform
3. Evaluate the prototype in terms of privacy, performance and usability to assess its suitability for a potential integration into the Threema app

This allows the Threema team to evaluate the feasibility of implementing such a system, which would provide them with insights into how clients use their app.

Chapter 2

Preliminaries

This chapter covers some concepts that are relevant to the project, which are necessary to understand the design and implementation of the solution.

2.1 Privacy Preserving Concepts

This section contains explanations for K-Anonymity and Differential Privacy, which will be referenced throughout the thesis.

2.1.1 K-Anonymity

K-anonymity is a technique to ensure that an individual’s data can “blend into the crowd” [NA21, p. 19]. The core idea is that every record in a dataset should be indistinguishable from at least $k - 1$ other records, so that no single person can be singled out based on their data.

To understand how this works, we have to look at *quasi-identifiers*. These are attributes that do not directly identify a person on their own, but can be used to do so when combined, such as age, postal code, or a combination of usage metrics. K-anonymity groups together all records that share the same quasi-identifier values, and a dataset satisfies k-anonymity if every such group contains at least k records [NA21, p. 19]. As a consequence, even an attacker who knows all the quasi-identifiers of a target will always be left with at least k possible matching records and cannot tell which one actually belongs to their target.

The parameter k controls the level of protection. A small k (for example $k = 2$) provides only weak protection, as a person blends in with just one other. A larger k means each individual hides among more people, which increases privacy but usually requires more data to be generalized or removed, reducing the accuracy and usefulness of the dataset.

To achieve k -anonymity, two main techniques are used: *generalization* [NA21, p. 21], and *removing outliers*, where particularly rare values are removed entirely [NA21, p. 23]. Both reduce how unique a record is, making it easier for individuals to blend into a larger group. Figure 2.1 illustrates this on a small example.

Original data		$\implies$	K-Anonymous data ($k = 3$)	
Age	ZIP code		Age	ZIP code
28	8001		20–39	800*
34	8004		20–39	800*
29	8008		20–39	800*
52	8702		50–59	870*
58	8706		50–59	870*
55	8703		50–59	870*

Figure 2.1: Example of k -anonymity with $k = 3$. In the original data, every record has a unique combination of quasi-identifiers (age and ZIP code) and could be singled out. After *generalization*, the records collapse into two groups (highlighted) that each contain three identical records, so any individual now blends in with at least two others.

2.1.2 Differential Privacy

Differential privacy is a mathematical approach to data privacy that allows insight about overall trends in a dataset without revealing any information specific to the individuals inside it [NA21, p. 8]. It is important to note that differential privacy is a property of the algorithms used to create the data [NA21, p. 27], not of the dataset itself. To show that a dataset satisfies differential privacy, you need to look at the algorithms that generated it and see if they meet the criteria for differential privacy.

Differential privacy guarantees that the output of a data analysis will be almost exactly the same with or without any single individual’s data included in the input. Because the final result doesn’t noticeably change whether your specific information is present or not, an attacker looking at the results will not be able to tell if you were part of the dataset, let alone figure out what your personal data is. [NA21, p. 27]

From the system’s point of view there is no inherent difference between a harmless query and a malicious attack designed to extract private data, since a query is merely an instruction for computation and its intent is not part of its syntax. The danger often lies not in a single suspicious query but in combining seemingly benign ones, for example asking for the sum of all ages and then the same sum with one individual removed already reveals that person’s exact age, even though neither query looks malicious on its own [NA21, pp. 16-17]. Differential privacy therefore makes no attempt to guess intent

and treats all queries uniformly. It adds enough noise to make the results of a malicious query entirely useless for identifying individuals, while keeping the noise small enough that the overall statistical trend remains accurate and useful for legitimate analysis.

To control the added noise, and thus the “amount of privacy”, differentially private algorithms expose a privacy parameter, usually written as the symbol ϵ and also called the privacy budget. It acts as a knob that bounds how much the algorithm’s output is allowed to differ between two datasets that differ only in the data of a single individual [NA21, p. 27].

A small ϵ enforces a tight bound, so the algorithm must produce nearly indistinguishable outputs in both cases. This means an observer cannot tell whether any given person’s data was part of the input, which requires more noise to be added and therefore provides stronger privacy. A larger ϵ relaxes the bound and lets the outputs differ more, which requires less noise and yields more accurate statistics, but offers weaker privacy [NA21, pp. 27-28].

2.2 Data Protection Regulation

Since we will be collecting data from users, data protection regulations might come into effect. Threema offers their service in both Switzerland and the EU, but also other parts of the World. Both Switzerland and the European Union have laws in place to protect the privacy and data of users. For simplicity and the scope of this thesis, we will consider data protection regulations for Switzerland and the EU only.

In Switzerland the Federal Act on Data Protection of 25 September 2020 (235.1) (FADP) and in the EU the General Data Protection Regulation of 27 April 2016 ((EU) 2016/679) (GDPR) is relevant to comply with data processing laws. Both laws come into effect when “Personal Data” is being collected or processed [FADP, Art. 2 para. 1][GDPR, Art. 3 para. 1.]. The definition on “Personal Data” varies slightly between the regulations, but is understood as any data that is relating to an identified person or can be used to identify a person[FADP, Art. 5 lit. a][GDPR, Art. 4 para. 1.].

An example for data that is relating to an identified person would be location data. An example for data that can be used to identify a person would be any data that is related to a person that has already identified themselves to the service.

While Threema allows users to stay totally anonymous, it is possible to provide ones phone number or email address, so others can contact you without knowing your ThreemaID. However, both the phone number and the email address are hashed and not readable by Threema, not making it a property where users are identified.

If we anonymize personal data and only collect data that cannot be linked to a person, we are exempt from data protection regulation. This means however that we have to

make sure data is truly anonymized. As the goal of this thesis is to implement a system that collects data in a way that no single person can be identified, we consider this criteria as already fulfilled

2.3 Data Submission Abuse Prevention

Whenever telemetry is transmitted to the Threema servers, an HTTP REST endpoint is called. This endpoint is public and therefore callable by everyone, even non-Threema users. In order to make sure the received data actually comes from devices that use Threema, we need some kind of authentication system that ensures no malicious data is submitted to the metrics server. Conventional authentication methods would not suffice in our case, since we have to make sure the authenticated devices stay anonymous and data cannot be associated with a person. For example, if we were to also transmit the ThreemaID by the user in order to authenticate the device as a valid Threema user, the data can be associated with this ID which in turn can be linked to real persons based on the email or phone number provided. This means that we need a system that is able to authenticate users by their valid ThreemaID without revealing their ID.

One well known system that enables this are zero-knowledge proofs. Zero-knowledge proofs allow a prover, in this case the user or device trying to send telemetry to the server, to convince another party, in this case the servers at Threema receiving the telemetry, without conveying any information beyond the mere fact of that statement's truth. [GMR85]. In our case, the statement would be 'I have a valid ThreemaID' and we would be able to prove that statement without conveying any information about our ThreemaID.

While this solves the problem of keeping your identity secret, the proof provided to the server would look identical every time for the same ThreemaID[TKLA23]. This would only mask our ThreemaID and would result in a problem called session correlation. Session correlation means we can identify the user over multiple sessions, and we could link data to that person.

There are cryptographic mechanisms to counter this problem for example nullifiers [GG22]. This would however exceed the scope of this thesis. We will instead fall back to IP based rate-limiting on the metrics server.

Chapter 3

Solution Design

In this chapter we will outline the process we have taken to select the metrics we want to collect, as well as the technical design of the telemetry system. It will form the basis for the implementation which will follow in the next chapter.

3.1 Implementation Platform

The first implementation of the telemetry system will be done for the Threema Desktop App, which is available for Windows, MacOS and Linux. We have chosen this platform because the application is electron-based, which offers the most familiarity for our team.

3.2 Metrics

Telemetry is only useful if the collected data can be used to answer questions we have about the app and its usage. Our goal is to provide a system which can be easily reused to track a variety of metrics. In order to do this, we first select metrics of different categories which are widely used and deemed relevant for the Threema app, then we implement them in a way that allows for easy reuse of the technical implementation for similar metrics.

3.2.1 Goals of the Telemetry

As specified in chapter 1, the goal of this project is to provide metrical evidence of app usage without compromising the privacy guarantees of the Threema product significantly. Telemetry could be used to support any of the following goals:

- ✓ **Usability (U):** Understand if and how features are used.
- ✓ **Performance (P):** Measure application performance in the real world.
- ✓ **Platform maintenance (M):** Understand under which circumstances apps are used and with which underlying platforms (OS, Devices).

The following goals will **not** be supported by telemetry collected in our project:

- ✗ **Stability:** The telemetry will not detect crashes or identify their causes, as this is already covered by Threema’s manually submittable crash reports and by the crash reports provided by the software vendors (Play Store, App Store) [Thr26a].

3.2.2 Criteria for Evaluating Metrics

In order to select the metrics we want to collect in our application, we will evaluate them as follows: We compare them to a list of constraints and requirements which they need to fulfill, then we check if there are possible obstacles with the technical implementation. We will select the final group of metrics in such a way that we have some diversity in both technical implementation as well as things which they measure. At the same time, the metrics should not reveal too much about any given user, which would increase the risk of re-identification. As preparation for the selection process, we have created a Threat Model (see Appendix A.4.2) to understand potential risks and attack vectors, as well as the actors involved.

Since privacy needs to remain a core value of Threema, the telemetry system must adhere to the strict constraints defined in Table 3.1.

Table 3.1: Constraints of the System

ID	Constraint
C-1	The implementation must not enable behavioral profiling of individual users.
C-2	The implementation must not provide per-user analytics or targeted personalization.
C-3	The system must be strictly opt-in with no additional incentive to enable it (additional features, ...).
C-4	No message content should be collected. The contents of messages or conversations cannot be inferred from collected data.

In compliance with the constraints and the goals defined in Section 3.2.1, the requirements in Table 3.2 can be derived.

Table 3.2: Requirements for metrics

ID	Requirement
R-1	Each collected metric has a clearly defined purpose.
R-2	The technical design must limit the potential for misuse by the data collector.
R-3	The metrics should contain no persistent identifiers.

While *C-2*, *C-3* and *C-4* are relatively straight-forward to verify for a given implementation, *C-1* is not. To combat this, we want to define some sort of property to measure how well a set of metrics can be used to profile and identify a singular user. Ideally, we would want to use a metric like *k-anonymity*, which is a widely used measure of anonymity. The problem we face with k-anonymity is that it is not calculable without already having access to the dataset, as would be our case. To overcome this, we will use Shannon entropy with bucketed values as a proxy for k-anonymity. This will allow us to estimate re-identification risk of a combination of metrics before having access to the dataset. This tries to ensure that the collected metrics are compliant with *C-1* while still allowing us to collect useful data for the Threema product team.

3.2.2.1 Setting a k-anonymity Target

While exactly determining a k-anonymity value before having data access is not possible, we will set a target value which we will then use to determine the maximum allowed entropy for our metrics. We assume all metrics to be sensitive values (quasi-identifiers) as described in the threat model. For this, we assume 12 Million Threema users as stated by the Threema Website[Thr26b]. We then have to assume a percentage of users who will opt-in to telemetry, which will determine the amount of users sending telemetry data. We reference a report of a analytics provider, which declares a 24% opt-in rate for developer analytics in their iOS app[OT22]. This opt-in rate is achieved with a one-time pop-up giving users the choice to opt-on or out of analytics, which is a standard approach for iOS analytics as done by Apple and was therefore deemed as representative for this thesis. To ensure some margin of error as well as accounting for the fact that Threema users may be more privacy-conscious than the average user, we round the percentage down to 20%. This results in the following assumed number of users sending telemetry data:

$$\text{Opt-in users} = 12'000'000 \cdot 0.20 = 2'400'000 \quad (3.1)$$

We also aim for a *K*-value of 10, which is stated as a conservative value in the literature[NA21, p. 2023].

3.2.2.2 Shannon Entropy as a Combination Budget

Shannon entropy H measures the average uncertainty of a metric's value distribution:

$$H(X) = - \sum_{i=1}^n p_i \log_2 p_i \quad (3.2)$$

where p_i is the probability of value i occurring. A high entropy value indicates many distinct, evenly distributed values, which increases re-identification risk on average. The relationship between entropy and k-anonymity results in a maximum tolerable entropy budget:

$$H_{\max} = \log_2 \left(\frac{N}{k} \right) = \log_2 \left(\frac{2'400'000}{10} \right) \approx 17.8 \text{ bits} \quad (3.3)$$

Figure 3.1 shows how $H_{\max}$ varies in respect to opt-in rates for selected k-anonymity targets, with the user base held at the assumed value of $N = 12'000'000$.

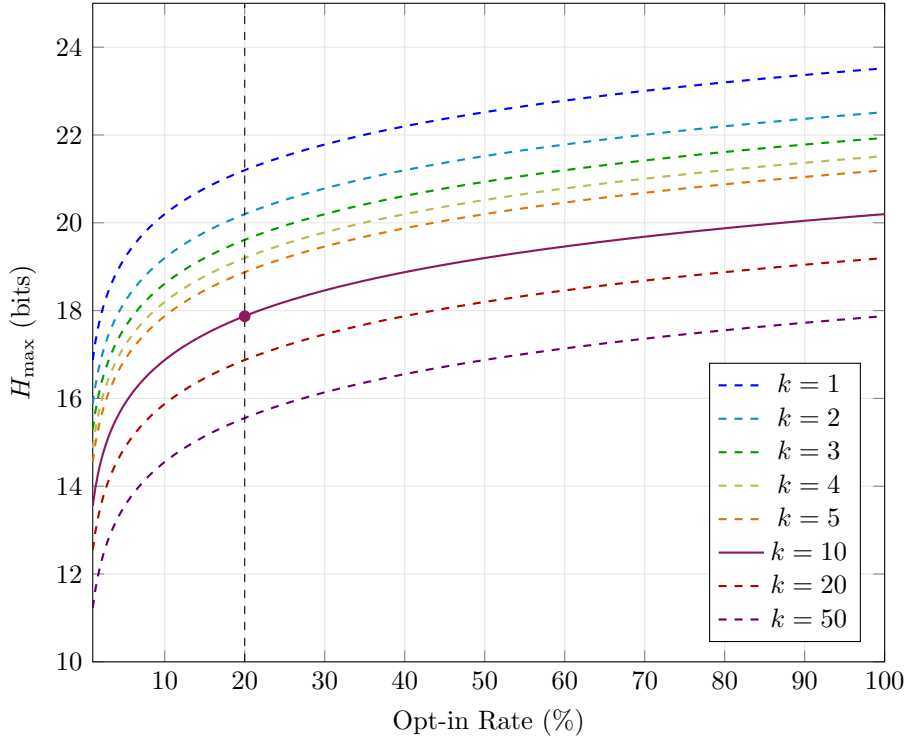


Figure 3.1: Maximum tolerable entropy budget $H_{\max}$ as a function of opt-in rate r for selected k-anonymity targets $k \in \{1, 2, 3, 4, 5, 10, 20, 50\}$, computed for a fixed user base of $N = 12'000'000$. The dashed vertical line marks the assumed opt-in rate of 20%, and the filled circle marks the assumed operating point ($r = 20\%$, $k = 10$), yielding $H_{\max} \approx 17.8$ bits.

Metrics can be conservatively combined by adding their individual entropies [Bez07, p. 3], which assumes independence. If two measured metrics are not independent from one another, the combined entropy may be less than the sum, but never more. In this case, the chance of re-identification would be lower. Any metric or combination of metrics whose combined entropy exceeds this budget risks producing groups where $k < 10$, violating our set anonymity requirement.

3.2.2.3 Entropy Control through Bucketing

For metrics with large or continuous value ranges, the entropy could in theory be unbounded, making it impossible to guarantee k -anonymity for even one metric. To combat this, we utilize *Bucketing*, which is done by grouping values into discrete buckets reducing the number of distinct possible values, while still providing useful numbers and distribution. This generalizes and clips data, two methods which are mentioned in the literature as effective ways to control k -anonymity [NA21, p. 21].

This means that, assuming buckets are chosen in a way that data is distributed evenly across them, the entropy of a singular metric M can be approximated:

$$H_M \approx \log_2(b) \quad (3.4)$$

where b is the number of buckets. This is the worst case entropy, where all the data is distributed evenly across the buckets. In case of uneven distribution, the entropy would be lower. In the presence of a metric with expected uneven distribution, it is also possible to calculate the entropy more precisely using Shannon’s formula (Equation 3.2) This would allow for a more accurate estimation of the total entropy of a combination of metrics.

In practice, this means we can freely decide the number of metrics and the bucketization of each metric, as long as the $\sum H_{M_i}$ of all metrics M_i is below the defined entropy budget $H_{\max}$.

3.2.2.4 Applicability of l -diversity and t -closeness

Two well known refinements of k -anonymity are l -diversity and t -closeness. Both were introduced to address two weaknesses of k -anonymity. The first refinement, l -diversity, requires that each group of indistinguishable records contains at least l well-represented values for the sensitive attribute. This protects against homogeneity attacks, where every member of a group shares the same sensitive value [GKM25]. By contrast, t -closeness goes further and requires that the distribution of the sensitive attribute within each group stays close, within a threshold t to its distribution in the overall dataset, protecting against attribute disclosure [Li25].

Both concepts are therefore relevant to consider, but they are not applicable to our system. The reason is that l -diversity and t -closeness both rely on a separation between quasi-identifiers (the attributes used to single out a record) and a distinct sensitive attribute (the value to be protected from disclosure once a group has been narrowed down).

Our model does not make this distinction: As stated above, we treat every collected metric as a sensitive value, so there is no separate sensitive attribute left to diversify or to balance against a global distribution. In other words, the threat we protect against is identity disclosure (re-identifying a user from a unique combination of metric values), not attribute disclosure within an already-formed group, which is the threat l -diversity and t -closeness are designed for. This matches the assets defined in our Threat Model (Appendix A.4.2), namely anonymity and unlinkability.

We therefore chose k -anonymity as our guiding concept for two reasons. First, it directly models our actual threat: A user being singled out by a rare combination of metric values. Second, none of these properties can be computed without access to the dataset. Of the three, k -anonymity is the one we can meaningfully approximate beforehand through the Shannon entropy budget described above. On the other hand, l -diversity and t -closeness would require knowledge of an in group sensitive attribute distribution that does not exist in our design.

3.2.2.5 Summary

With our established model, we can experiment with different combination of metrics and bucketization strategies to find a set of metrics that provides useful insights into app usage while maintaining a low re-identification risk. For example, with our defined entropy budget of $H_{\max} \approx 17.8$ bits, we could choose to collect 8 metrics with 4 buckets each, which would result in a total entropy of $H = 8 \cdot \log_2(4) = 16$ bits. Another example could be collecting 3 metrics to a very granular level of 60 buckets each, which would result in a total entropy of $H = 3 \cdot \log_2(60) \approx 17.7$ bits.

While we have determined a maximum entropy budget of $H_{\max} \approx 17.8$ bits, this is only an example and changing any of the assumptions as listed in Table 3.3 would change this budget. For example, if an application of similar user counts to Threema would make metrics mandatory instead of opt-in and decreasing the target k -value to 5, the budget would increase to $H_{\max} \approx 21.2$ bits, which would allow for more metrics or more granular bucketization.

Table 3.3: Assumptions made to calculate entropy budget.

Subject	Assumption	Conservativeness
Threema User Base	12'000'000[Thr26b]	Neutral
Opt-in Rates	24%[OT22], rounded to 20%	Conservatively rounded
k -anonymity Target	$k = 10$ [NA21, p. 2023]	Conservative
Correlation between Metrics	No correlation	Worst-case assumption
Bucket Distribution	Evenly distributed	Best-case assumption

It is important to understand that while our framework provides a way to approximate a combination of metrics which are compliant with the set constraints, it is impossible to guarantee any k-anonymity value before having access to the dataset. Even though we made conservative assumptions throughout our calculations, it is still possible that a user with extraordinary usage patterns or a poor bucketization strategy, could be identified in the dataset. This is why we recommend verification of the collected user data once it is available and open communication with the user base about the collected data.

3.2.3 Metrics Selection

We grouped the candidate metrics into eight categories, each grounded in established literature. **Acquisition & Growth** and **Retention** follow the widely used *AARRR* (Acquisition, Activation, Retention, Referral, Revenue) framework introduced by McClure [McC07]. **Engagement** captures how intensively users interact with the application, a construct that is commonly measured through behavioral metrics such as session frequency, duration and stickiness [LLYTD12]. **Communication** reflects the messaging behavior that is central to an instant-messaging service, which prior work has characterized by analyzing real-world messaging behavior at scale [SPHS22]. **Feature Adoption** measures which application features are actually used, a standard dimension of product analytics [Amp26b]. **Technical / Performance** covers application performance as experienced by users, an established determinant of mobile Quality of Experience [DMW20]. **Social Graph** concerns the size and structure of a user’s contact network, which has been measured directly from the social graphs of online platforms [MMG⁺07]. **Notification & Reachability** addresses how and when users attend to incoming messages and notifications [PdOKO14]. Each category links to the corresponding table in the **appendix**, which lists its metrics together with the source establishing each one.

While we deem none of these categories impossible or inappropriate to implement on their own, we will narrow down our selection because the project scope allows only a limited number of metrics to be implemented:

- The *Acquisition & Growth* category is excluded as Threema account creation is not supported on the desktop platform.
- The *Notification & Reachability* category is excluded because it would require a implementation on mobile devices, while our first implementation will be on desktop. It would also likely require some connection with the push service, which would increase complexity.

For the remaining six categories, we pick one metric each to implement, while keeping the overall selection small and technically diverse.

Following Threema’s suggestion, we have compared three widely used applications for metric monitoring, namely Amplitude, Mixpanel and Fullstory and have created a table of the features they provide, the summary can be found in the **appendix A.2**.

3.2.3.1 Selected Metrics

In exchange with the Threema Team, we select one metric of each category based on two criteria: The metric must be deemed to provide data which is interesting for analytics. It also should be an implementation which is built in such a way that it can possibly be reused for tracking of other, technically similar metrics. The selected metrics and the rationale behind measuring them are listed below:

Engagement: Stickyness

Question: How frequently is the desktop app used over a defined time frame?

Motivation: Distinguish daily usage from occasional usage and evaluate whether the desktop app is used as a primary client or not.

Definition: Number of distinct calendar days the application is opened within the reporting window.

Engagement: User Resurrection

Question: After a period of inactivity, how long do users take to return to the desktop app?

Motivation: Detect whether the desktop app is used continuously or in bursts, and evaluate the impact of changes geared towards re-engagement (e.g., usability improvements, onboarding hints, performance fixes).

Definition: Time between the end of the last active session and the start of the next active session.

Communication: Message Type Distribution

Question: What types of messages are sent from the desktop app within the reporting window?

Motivation: Understand which communication modalities are most relevant on desktop to prioritize UI/UX and performance.

Definition: Ratios of successfully sent messages per message type within the reporting window.

Feature Adoption: Use of "Copy Poll" Feature

Question: How often is the "Copy Poll" feature used on desktop?

Motivation: Decide whether support and maintenance for the feature should be continued and improved.

Definition: Count of "Copy Poll" actions within the reporting window.

Technical / Performance: App load time

Question: How fast does the app start on different desktop environments?

Motivation: Identify whether users on slower machines or specific platforms experience poor startup performance and prioritize performance work accordingly.

Definition: Time from process start until the UI is responsive, measured locally.

Social Graph: Active Contacts

Question: With how many contacts did the user communicate during a set period of time?

Motivation: Inform UI scaling decisions (search, list rendering, navigation) and help prioritize optimizations for small contact lists versus power users with large address books.

Definition: Number of contacts with which the user communicated directly or via group-chats.

3.2.3.2 Summary

We selected six metrics from all of the categories we have evaluated as feasible and relevant for Threema Desktop. Together they cover how often the app is used, what users do in it, and how well it runs. They also offer a diverse set of technical challenges, which will result in a product which can be used to track a wide variety of metrics with similar technical requirements in the future.

3.3 Random Sampling

As explained in Section 3.2.2, when implementing our metrics system in Threema, our privacy budget would allow for approximately 8 metrics with 4 buckets each. While this is sufficient for a proof-of-concept, a system built for the real world would likely measure more metrics in order to collect data on more features or management ratios. To achieve this, *Random Sampling* could be used, where in a larger set of metrics, only a selected few are sent.

3.3.1 Random Metric Selection

To select the metrics to be sent, we have identified the following methods:

1. All users report the same metrics for each reporting window, changing measured metrics each time.
2. For each reporting window, a random subset of metrics is selected to be sent.
3. Each user reports a random subset of metrics for the whole time, meaning that the same metrics are always reported by the same user.

We will now evaluate how these approaches would affect privacy in the following sections.

3.3.1.1 All users report the same metrics

The **first approach** would not affect our privacy budget, as the same metric for each month is reported by all users. This approach would leave holes in the data, as a metric is only reported on a certain month. These gaps grow bigger as more possible metrics are added. However, it supports measuring a theoretically infinite amount of metrics, although this would take an infinite amount of time.

3.3.1.2 Random subset of metrics per reporting window

For the **second approach**, entropy for this could be calculated using the Equation 3.2, but with an added term for probability p of the metric being reported, and b being the number of buckets. All other assumptions still hold as well.

This would allow measuring of many different metrics at the same time, but with a lower total number of users reporting a specific metric.

In practice, we would use a weighted random sampling approach, where the probability of reporting a metric is higher for more important metrics, and lower for less important metrics. Then, we can fill the privacy budget using metrics with weighted selection until the budget is filled. This avoids having users which report very few or too many metrics due to the randomness of the selection. A notable advantage of this approach is that it introduces an additional layer of uncertainty for an attacker: Even if they have access to the metrics database as well as external information about a user, they cannot be certain about which metrics a user reported. This means that even if they are able to single out a user, they cannot be certain that this user is the one they are targeting, because their target might not have reported the known metrics at all.

3.3.1.3 Fixed random subset of metrics per user

The **third option** is similar to option two, but has one notable disadvantage. Since a client would report the same metric for his whole lifetime, this information can be used to help with re-identification by adding a second component to the entropy: $H(S)$ describes the entropy of the subset which the user reports. This entropy grows with the number of total possible metrics, meaning re-identification would become easier as more metrics are added to the system.

3.3.1.4 Conclusion

After considering all possible methods, we have decided that method 2 (Each user reports a randomly selected subset of metrics) provides the most coverage while also decreasing re-identification risk.

3.3.2 K-anonymity under Random Sampling

Our chosen method of random sampling as described in the previous section results in each user only reporting a subset of the possible metrics, effectively resulting in many missing values for each user. If we were to use a traditional approach to calculating k-anonymity by treating a non-reported metric as a value, the k-anonymity of the dataset would be unrealistically low. This would be because the non-reported metrics add additional calculated entropy to the dataset. However, because we assume that non-reported metrics cannot be predicted or otherwise externally extracted by an attacker, this entropy would not contain useful data for re-identification. Therefore, we try to find a k-anonymity calculation which produces more meaningful values.

To do this, Ciglic et al. [CEK14] suggests the following methods:

1. **Basic match:** NULL values match with nothing (not even NULL) → rows with NULL are removed.
2. **Extended match:** NULL values match only with NULL → NULL is treated like a regular value.
3. **Maybe match:** NULL values match with any value → NULL act as wildcard.

Additionally, we have identified another method based on our threat model (Appendix A.4.2), which assumes that an attacker has external knowledge about a subset of the metrics for a user. This method is described as follows:

4. **K-anonymity on feature subsets:** In this approach, we would calculate k-anonymity for each possible combination of N metric types. This would reflect the fact that an attacker would not know all metric values for a user, and has to rely on a subset of metrics for re-identification. The k-anonymity value could then be calculated as the average or minimum across all metric type combinations.

To evaluate these methods, we calculate them alongside the traditional k-anonymity calculation in a test scenario from zero to 5000 users. We generate metric batches as they would be generated in our package, then continuously evaluate the k-anonymity of the dataset using the different methods. The results are shown in Appendix A.5.2.

We choose the subset method as the most meaningful method, as it reflects the assumptions of our threat model the best and provides meaningful k-anonymity values. As an added bonus, it gives us two different k-anonymity values to evaluate: The *average k-anonymity* across all subsets (how protected is the average system user), and the *minimum k-anonymity* across all subsets (more akin to a traditional k-anonymity value, which reflects how protected the most vulnerable user is).

Because the subset method can produce varying k-anonymity values depending on the subset size, we also want to fix a subset size for the main k-anonymity value we will use for future evaluation. Taking our threat model (Appendix A.4.2) into consideration, **we choose a subset size of 3 as a default**, as we deem this a reasonable upper bound for the number of metrics an attacker would be able to predict for a user.

However, we would like to note that the target subset size is not fixed and can be adjusted by the operator of the telemetry system if they wish to do so based on their own assumptions.

3.3.3 Formulas for k-anonymity

The main goal of this system is to collect metrics in a way that minimizes the risk of re-identification for users. Here, we consider two key numbers:

- The average risk of re-identification for users in the system
- The worst-case risk of re-identification for a user in the system

We want to deliver a way to calculate the expected values for these two key numbers for any combination of system parameters (e.g. number of users in the system, number of metrics, number of buckets per metrics, etc.). For this, we have derived formulas for the expected average k-anonymity (baseline k-anonymity) and the expected minimum k-anonymity.

To calculate the **average group size** k_{base} , we can use the following formula:

$$k_{base} = \frac{n_B \cdot \binom{M_R}{M_S}}{b^{M_S} \cdot \binom{M_T}{M_S}} \quad (3.5)$$

where

- n_B is the number of batches in the system for the reporting period
- b is the number of buckets per metric
- M_R is the average number of reported metrics per batch
- M_S is the size of the metric subset being used for k-anonymity calculation
- M_T is the total number of available metric types

This formula is derived from equations 3.3 and 3.4, with the added division by $\binom{M_T}{M_R}$ to account for the fact that each batch only reports a subset of the available metrics and therefore can only be compared to other batches in this subset. The goal of this formula is to provide an aim point for the measured average k-anonymity values, because it assumes that the buckets are distributed evenly. If the measured average k-anonymity values are significantly higher than this formula, it would indicate that the buckets are not distributed evenly.

To calculate the **expected minimum k-anonymity value** k_{min} , we can use the following formula:

$$\begin{aligned} \mu &= \frac{n_B \cdot p_{\min} \cdot \binom{M_R}{M_S}}{\binom{M_T}{M_S}} \\ N &= c_{\min} \cdot \binom{M_T}{M_S} \\ k_{\min} &\approx \mu - \sqrt{2\mu \ln(N)} + \frac{\ln(N)}{2} \end{aligned} \quad (3.6)$$

where

- n_B, M_R, M_T, M_S are defined as above
- $p_{\min}$ is the probability of the least likely combination of M_S metric values, which can be calculated by multiplying the probabilities of the M_R least likely buckets over different metric types.

- $c_{\min}$ is the number of combinations of metric values that have the probability $p_{\min}$, which can be calculated by multiplying the number of buckets with the least likely probability across different metric types.

For example, if all metrics would have the same number of buckets and the same distribution, this would be $p_{\min} = (\frac{1}{b})^{M_S}$ and $c_{\min} = b^{M_S}$. As buckets of different metric types get more unbalanced, the value of $p_{\min}$ gets smaller, and therefore the expected minimum k-anonymity value in the rarest bucket gets smaller as well. Further details on the derivation of this formula can be found in Appendix A.6.2.

3.4 Dynamic Privacy Budget

In a real deployment, the metrics collection would roll out gradually due to clients being implemented one after the other, as well as users opting-in at different times. This would mean that potential early adopters would not be as protected from re-identification as later adopters, because the effective privacy budget would be much smaller. To mitigate this issue, we scale up how many metrics are measured in such a way that the privacy budget is never exceeded. We achieve this by publishing the privacy budget $H_{\max}$ on an endpoint, which will allow clients to get it and then only submit enough metrics so that $\sum H_{M_i} \leq H_{\max}$. As more users opt-in, this privacy budget will increase, allowing clients to send more data.

To calculate and provide the privacy budget, at first we have implemented it as a simple function of the number of opt-in users and desired k-anonymity, as shown in Equation 3.3. This method provides budget in such a way that the k-anonymity of the average user does not fall below the target k-anonymity.¹ However, this **"Simple"** mode does not take into account the random sampling of metrics, which introduces null values and thus makes measuring a traditional k-anonymity difficult. As discussed previously, we have adapted the k-anonymity measurement by introducing subset k-anonymity. While this allows us to measure k-anonymity again, it means that the target k-anonymity value does not match with the measured k-anonymity, because they are based on different measurement methods. Additionally, it does not take extra steps to shield the least protected users, which means that the measured $k_{\min}$ can be much lower than the target k-anonymity.

To combat this, we introduced two additional modes: **"Average k"** and **"Minimum k"** mode. These modes optimize for Equations 3.5 and 3.6, respectively, by adjusting the privacy budget based on the predicted k-anonymity values. We have evaluated these modes in Appendix A.5.4, and found that they ensure that the measured subset k-anonymity is above the set target. However, because this newly defined methods optimize for this subset k-anonymity, actually favors more metrics submitted, which leads to a higher measured k-anonymity due to more combinations being available, as explained in the Appendix.

¹Tested in Appendix A.5.1

Because of this, we have decided to recommend the **"Simple"** mode, which is more conservative and does not rely on the subset k-anonymity measurement. The simple mode requires the least amount of assumptions and is a robust, easy to understand method. We have verified its functionality and that it provides the expected k-anonymity on average without random sampling. Adding random sampling to the system only increases privacy because it adds more uncertainty to the data, so while direct performance of the simple mode with an equivalent k-anonymity measurement is not possible, the overall privacy is guaranteed to be met. The formula can also be adjusted to be more conservative, aiming for the minimum k-anonymity instead of the average. The other modes are still available in the final product for evaluation and the subset k-anonymity measurement provides a useful tool to measure the k-anonymity of the collected data.

3.5 Opt-In

In this section we will outline the design of the opt-in system, which is a crucial part of our implementation.

3.5.1 Consent Rate vs Opt-in Rate

It is important to understand the difference between the consent rate and the opt-in rate, as they are often used interchangeably but have distinct meanings.

The **Consent Rate** measures the proportion of positive consents relative to the total number of choices made by users. It focuses on the users who actively interacted with the consent notice and made a decision [Did24].

$$\text{Consent Rate} = \frac{\text{Opt-ins}}{\text{Total Choices}} \quad (3.7)$$

In contrast, the **Opt-in Rate** measures the percentage of users who responded positively out of all users who were shown the consent notice. This metric takes into account every time the notice was displayed, regardless of whether the user interacted with it or not [Did24].

$$\text{Opt-in Rate} = \frac{\text{Opt-ins}}{\text{Total Notices Displayed}} \quad (3.8)$$

To maximize the consent rate, it is important to explain to the user why they should opt-in and what benefits the telemetry provides, while to maximize the opt-in rate, it is crucial to ensure that the notice is seen by as many users as possible and that it is not overseen without interaction.

3.5.2 Placement

For the placement of the opt-in, popups are commonly used. For opting out, after accepting the popup, users go to the settings and search for a section related to privacy

or data collection, where they can then disable telemetry using a checkbox.

3.5.2.1 Settings

A common placement of the checkbox is in the settings menu, under a section related to privacy or data collection. This allows users to easily find and change their preferences regarding telemetry, while also keeping the option to opt-in accessible at all times. However, a blog post on opt-in telemetry from GNOME suggests that relying solely on a settings checkbox may not be sufficient, as users often do not explore settings menu [Tes23]. Additionally, the fact that not many people explore the settings menu could point to a bias in the collected data, as only users who are more engaged with the app would be represented in the dataset, which could lead to a wrong understanding of the user base. This brings us to the use of a popup, which is a more direct way to ask users and also increases the opt-in rate but not necessarily the consent rate, we will tackle this later.

3.5.2.2 Popup Design

Designing an effective opt-in popup is crucial for maximizing the consent rates. According to CommerceGurus[Com24], the design of the pop up should enable users to make an informed decision without feeling pressured or manipulated.

Key considerations for the popup design include:

Visual Integration: The popup should seamlessly integrate with the application’s overall design language. A native-looking dialog creates trust, whereas a different design might be perceived as intrusive or suspicious.

Description of Value: The text must explain the benefits of opting in, specifically, how the data contributes to the stability and improvement of the software, without resorting to vague marketing language or buzzwords. This bachelor thesis itself can be used as a reference to provide some transparency behind the process of the implementation.

Simplicity and Control: The interface should be uncluttered, presenting a binary choice that is easy to understand [Comnda]. The design of the consent and denial buttons should reflect the brand’s values. If the brand prioritizes transparency and wants to avoid dark patterns, both options should be equally prominent and visually distinct [Comnda]. In contrast, if the goal is to maximize opt-in rates, the consent button would be designed to be more visually appealing while the denial option is made less prominent. However, staying true to the brand’s values is very important according to CommerceGurus [Comnda].

We chose to ask Threema directly about the design and they suggested using a primary and secondary button. With this in mind we designed three options from which they

could choose, shown in the appendix [A.7.1](#). They decided on Option 3, yes button as primary and no button as secondary.

Triggers: Website implementations might employ “exit-intent” triggers, which according to CommerceGurus [[Comndb](#)] are crucial. The reasoning is that visitors typically spend only a short amount of time on a website, so the popup has to capture them before they leave. An application like Threema, by contrast, is used regularly and over a long period, so there is no need to interrupt the user from closing the app. The desktop environment would therefore profit from displaying the popup at a non-intrusive moment, such as upon application startup or in the onboarding process.

3.5.3 Summary

In this section, we established the distinction between consent rate and opt-in rate. To achieve a high opt-in rate we decided against hiding the option solely in the settings menu, which often leads to low engagement and potential selection bias. Instead, we favor a well-integrated popup, displayed during application startup or in the onboarding process for maximizing the opt-in rate and a transparent description with a potential blog post explaining our work to increase the consent rate.

3.6 Testing Concept

The goal of the evaluation is to determine whether the proposed telemetry system achieves its intended privacy guarantees while remaining practical to use. Since the system is implemented as a proof of concept, the focus is not on demonstrating production readiness but on validating the underlying design decisions and assessing their strengths and limitations.

The evaluation aims to answer three main questions:

- Does the system provide privacy by design?
- Does the implementation correctly enforce the privacy mechanisms?
- Are there practical factors that could weaken the anonymity of users despite the intended design?

To answer these questions, a combination of theoretical analysis and practical testing is used.

3.6.1 Testing Methods

To evaluate the proposed telemetry system, a combination of simulation-based testing, unit testing, and network testing is used. Synthetic datasets will be used to evaluate the privacy properties of the system and to validate the anonymity models developed in this

thesis. These experiments provide insight into how the system behaves under different conditions and allow the theoretical predictions to be compared with observed results.

Unit tests are used to verify that the client-side implementation correctly enforces the intended privacy-preserving mechanisms. In addition, network testing will be performed to identify potential information leaks that may exist despite transport encryption. In particular, these tests will investigate whether observable network characteristics, such as packet sizes, could reveal information about telemetry submissions and thereby weaken user anonymity.

3.6.2 Summary

Together, these evaluation methods provide evidence for both the theoretical soundness of the proposed design and the correctness of its implementation. The results of these evaluations are presented in Section 5.1.

Chapter 4

System Development

This chapter describes the development of the solution, including the functional requirements for our prototype, the design decisions made, technologies used and the implementation details.

4.1 Functional Requirements

This section defines the functional requirements for the prototype. They describe what the system has to do, whereas the privacy constraints (Table 3.1) and the metric requirements (Table 3.2) defined in the solution design restrict how it is allowed to do it.

The system is split into three parts: a client-side metrics package that is embedded into the Threema Desktop application, a metrics server that receives and aggregates the collected data, and an administrative interface that allows for monitoring and managing the system. The requirements are therefore grouped accordingly.

Each requirement is prioritized using the Must have, Should have, Could have, Won't have (MoSCoW) scheme [AAK⁺24] in order to distinguish what is essential for a viable prototype from what is merely desirable. The priority is encoded by color throughout this section, as described in Table 4.1.

Table 4.1: MoSCoW prioritization legend

Priority	Meaning
Must	Essential for the prototype.
Should	Important but not vital, not essential for the prototype.
Could	Desirable, implemented only if time and scope permit.

4.1.1 Threema App Client Requirements

The client requirements, as shown in Table 4.2, describe the behavior of the metrics package that runs inside the Threema Desktop application. They cover obtaining user consent, collecting and storing data locally, applying the privacy mechanisms and transmitting the result to the metrics server.

Table 4.2: Functional requirements of the Threema client

ID	Requirement	Priority
FR-C01	The application presents the user with an opt-in dialog.	Must
FR-C02	The users opt-in decision is persisted and respected across application restarts.	Must
FR-C03	The user can change their decision and opt out again at any time through the application settings.	Must
FR-C04	No telemetry is collected or transmitted unless the user has explicitly opted in.	Must
FR-C05	The package collects user interaction events by listening for actions on tagged user-interface elements.	Must
FR-C06	Collected data is stored locally on the device until it is transmitted.	Must
FR-C07	Each metric value is mapped into a discrete bucket before it leaves the device.	Must
FR-C08	Differential privacy noise is applied to the bucket index.	Must
FR-C09	Differential privacy can be enabled or disabled per metric.	Must
FR-C10	Metrics are transmitted in batches without any persistent identifier and with padding to obscure the batch length.	Must
FR-C11	Telemetry is transmitted in a reporting window.	Must
FR-C12	The metrics only get sent if enough users have opted in.	Must
FR-C13	The set of transmitted metrics is selected based on the available entropy budget and the metric weights provided by the server.	Should
FR-C14	The collected metrics and the privacy and transmission parameters are configurable without changing the application code.	Should
FR-C15	The user can inspect the telemetry data that is stored locally on their device.	Could
FR-C16	The setting gets persisted across platforms.	Could

4.1.2 Server Requirements

The server requirements, shown in Table 4.3, describe the behavior of the backend that receives the batches transmitted by the clients, protects the endpoint against abuse and makes the aggregated data available for analysis.

Table 4.3: Functional requirements of the server

ID	Requirement	Priority
FR-S1	The server exposes an endpoint that accepts telemetry batches submitted by clients over an encrypted connection.	Must
FR-S2	The server stores the received telemetry.	Must
FR-S3	The server does not persist the IP addresses of submitting clients.	Must
FR-S4	The server limits the rate of submissions per origin in order to mitigate abuse of the public endpoint.	Could

4.1.3 Administration Tool Requirements

The administration tooling consists of a configuration frontend and a set of dashboards. The configuration frontend controls the privacy and collection parameters that are propagated to the clients. The dashboards turn the aggregated telemetry into insights the product team can act on. The requirements for these tools are listed in Table 4.4. These tools are internal and are not exposed to end users.

Table 4.4: Functional requirements of the administration tooling

ID	Requirement	Priority
FR-A1	Administrators can configure the target k-anonymity.	Must
FR-A2	Administrators can suggest a weight that influences the probability of a metric being sent.	Must
FR-A3	Administrators can set an interval in minutes for recalculation of privacy budget and other parameters.	Must
FR-A4	A dashboard that visualizes the aggregated metrics, including their distribution across buckets and their development over time.	Must
FR-A5	A dashboard that shows the history of configuration changes, such as the target k-anonymity and the interval.	Should
FR-A6	A dashboard that monitors the volume of received send requests and metric batches.	Should
FR-A7	A dashboard that visualizes the correlation between different metrics and how it develops over time.	Could
FR-A8	A dashboard that visualizes the different k anonymities of the collected metrics and their history.	Could

4.2 Market Study

In this section we will have a look at existing solutions and products that might cover our requirements already.

4.2.1 Criteria Fulfillment

Our solution must have the following abilities:

- Apply differential privacy algorithms on the client side data.
- Integrate into the existing Threema Desktop app as a node package.
- Batch send telemetry data by accumulating data over a time window and transmit.
- Universal interface to access data from the backend server.
- Open source and free license permitting altering software for commercial use

4.2.2 Existing Tools

After a thorough examination on existing tools, we have identified the following:

- **PostHog:** An open-source product analytics platform that supports event tracking, session recording, and feature flags. It can be self-hosted or used as a cloud service and provides SDKs for Node.js environments [Pos26].
- **Aptabase:** A privacy-focused analytics platform purpose-built for desktop and mobile applications, including explicit support for Electron. It offers a lightweight event-based API and can be self-hosted [Apt26].
- **Mozilla Glean:** A telemetry SDK developed by Mozilla, used in production in Firefox and other Mozilla products. It provides typed metric definitions via a YAML schema, built-in ping scheduling, and a Node.js package, though its Node.js implementation lacks persistent storage across sessions [Moz26].
- **TelemetryDeck:** A privacy-focused analytics service that aggregates telemetry server-side and provides a dashboard for data visualization. It offers an SDK for web and Node.js environments but does not support client-side differential privacy [Tel26].

In table 4.5 we compare our selection of existing tools against the requirements we must meet. We can see that no existing tool matches all requirements. None of the tools implement a differential privacy algorithm, which we defined as a requirement to meet the privacy constraints. No existing solution would satisfy the requirements with only minor additions and changes to the existing code. Implementing our own system might come at a more development effort, but it allows greater freedom of development for the specific use case for Threema and comes with less technical debt. We therefore come to the conclusion that we implement our own data collection system.

Table 4.5: Comparison of existing telemetry tools against requirements

Requirement	PostHog	Aptabase	Glean	TelemetryDeck
Client-side differential privacy	×	×	×	×
Node.js / Electron integration	✓	✓	×	✓
Batch telemetry transmission	×	×	×	×
Universal backend interface	✓	✓	~	×
Open source / Free license	✓	✓	✓	✓

Note. ✓ = fully supported; ~ = partially supported; × = not supported.

4.3 Data Transmission

Collected metric data has to be sent from a Threema Client to the metrics server, providing the ability to link metrics and therefore allowing for more useful insights into application use, while also not providing a way to identify the user which sent the set of metrics. To achieve this, we will be sending metrics in batches with no additional persistent identifier. Such a metrics batch will include the following:

1. Collected metrics since the last metrics transmission in key-value pairs or other fitting data structure.
2. Padding to reduce the ability of an attacker to interpret message length as a way to gain information about the user.

For transmitting the data, we will be using HTTP as it is commonly used, has wide cross platform support and contains built-in transmission encryption. Due to the nature of the HTTP protocol, the Threema Server will always be able to observe IP address of the client sending the metrics. This could in theory be used to link multiple batches of data together, which is why the metrics server will not persist any IP addresses. Furthermore, in practice an IP address may fluctuate over time and steps can be taken by cautious users to change their IP address. While there are ways to hide the sender IP-address from the server (e.g. the Tor Network), all of them come with considerable complexity for both implementation and maintenance. In consideration of these factors, we have decided to accept HTTP request with TLS as a way to transmit data between client and server.

4.3.1 Defining Bucket Ranges

Our implementation uses metrics bucketing as a key tool for avoiding user re-identification. Bucket ranges must be carefully defined for each metric, as they determine how continuous values are discretized before differential privacy mechanisms are applied.

Initially, we considered implementing a dynamic bucketing approach where clients would query current bucket ranges from the server before submitting metrics. This approach offers several advantages: Bucket ranges could be adjusted post-launch if the system determines that buckets are significantly out of balance, allowing the anonymity guarantees to be maintained without requiring client updates. Additionally, it would eliminate versioning inconsistencies that could arise from users running different client versions, which might otherwise submit different bucket values for the same measured metric. However, this approach introduces a considerable security risk: A malicious metrics collector could adjust bucket ranges in a way that effectively re-identifies users. For instance, an attacker could make certain buckets extremely small, causing only very few users to fall into these buckets, which increases re-identification risk for these users.

Given these considerations, we faced a trade-off: Should we optimize the system for privacy under the assumption of a trusted server, or should we design for worst-case scenarios where the server may be malicious? We have decided to implement **static bucket ranges** that are embedded within client versions and cannot be frequently changed. This design decision prioritizes robust privacy guarantees in adversarial settings over operational flexibility. While this approach necessitates client updates to adjust bucket ranges post-launch, it prevents the server from manipulating bucket distributions to compromise user privacy.

A remaining challenge with static buckets is that different client versions may continue to submit different bucket values for the same measured metric value, potentially skewing aggregate data. This limitation is accepted as a necessary trade-off to maintain privacy guarantees under adversarial conditions.

4.3.2 Visualization

To summarize how a single measured value is transformed before it reaches the backend, Figure 4.1 traces the “Copy Poll Usage” metric through the pipeline. Each line represents one user’s measured value, that is, the number of “Copy Poll Usage” actions counted within the reporting window. The raw counts (left) are first *bucketed*, which assigns each value to one of four discrete buckets, each represented by a color, as shown in the legend of Figure 4.1.

Differential privacy noise is then applied to the bucketed values. The result after differential privacy (right) shows how this noise can shift a value into a neighboring bucket, which is why some records change color between the bucketed and the final column.

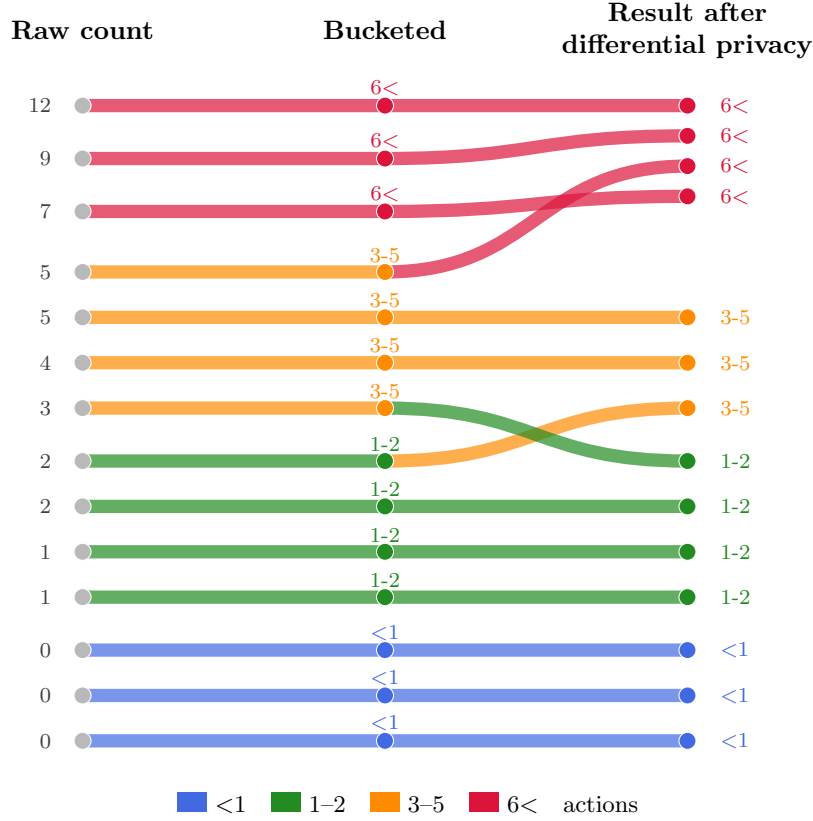


Figure 4.1: Alluvial diagram of the “Copy Poll” metric pipeline. The colored key denotes the four buckets a raw count is assigned to.

4.4 Technology Decisions

The following section describes the technology decisions we made for the implementation of our solution.

4.4.1 Metrics Package

As mentioned, the goal of this Bachelor thesis is to implement privacy friendly metrics specifically on the Threema desktop client as a proof of concept. The Threema desktop application is built using Electron, which makes it possible to create a node package for the metrics implementation and integrate it into the existing codebase. This would make it modular and decoupled from the rest of the codebase.

The implementation in a node package does not work in the mobile clients, which are implemented in Kotlin and Swift and will have to be rewritten accordingly if the system is to be expanded to mobile clients.

4.4.2 Metrics Server

The implementation of the metrics server will be done in Python. While Rust is used in other parts of the Threema ecosystem, we chose Python for the server-side component for several reasons:

- **Data Analytics Ecosystem:** Python is the de-facto standard language for data science and analytics. Libraries like Pandas and NumPy provide robust tools for data manipulation and analysis that are not as mature or readily available in the Rust ecosystem. Since the primary purpose of this server is to aggregate and process metrics, leveraging these existing tools significantly accelerates development.
- **Performance Trade-offs:** For the computational parts, Python libraries like Pandas are built on top of C, offering performance that is sufficient for our use case.
- **Analytics Tooling:** The visualization of the collected data is handled by Grafana, as described in Section 4.4.4. However, should one not want to rely on it, Python's ecosystem includes powerful visualization libraries like Pandas, Matplotlib, and Seaborn, which allow for quick visualization and reporting directly from the server.

4.4.3 Metrics Database

For the storage of the metric data, we evaluated three options:

- **Plain PostgreSQL:** It handles the relational aspect of our data model well (e.g., joining 'Metric' with 'MetricType').
- **InfluxDB:** A dedicated Time Series Database (TSDB) optimized for high write throughput and storage efficiency. It uses a specific query language (Flux/InfluxQL) and lacks full relational support.
- **PostgreSQL with TimescaleDB:** TimescaleDB is an extension for PostgreSQL that optimizes it for time-series data using "hypertables" (automatic partitioning). It allows use of standard SQL, maintain foreign key constraints between metrics and their types, and benefit from TSDB performance features like compression and continuous aggregates.

Decision: We chose **plain PostgreSQL**. The deciding factor was that it handles the relational aspect of our data model well, in particular the joins between the 'Metric' and 'MetricType' tables, while keeping the setup simple and using only standard SQL. For the scale of the prototype, the relational capabilities of PostgreSQL are sufficient and the dedicated time-series features of a TSDB are not required for a prototype.

Because TimescaleDB is an extension of PostgreSQL, it can be enabled on the existing database and our standard-SQL queries continue to work unchanged. Taking full advantage of its time-series features would, however, require converting the relevant tables

into hypertables, which entails some schema adjustments, most notably including the time column in their primary keys. Should the collected data grow enough for these performance features to become worthwhile, this migration remains realistic, which is why we list it as future work (Section 6.2).

4.4.4 Visualization

The administration tooling has to visualize the aggregated metrics and the privacy parameters for the Threema team (requirements [FR-A4–FR-A7](#)). We considered building these visualizations directly into the administration web frontend, but doing so would have meant implementing and maintaining querying, charting and time-series handling ourselves.

Instead, we chose Grafana, an open-source analytics and visualization platform. Grafana connects directly to our PostgreSQL database as a data source and lets us express each panel as a plain SQL query, which fits our relational data model and avoids copying the data into a separate analytics store. Both the data source and the dashboards can be defined as code and provisioned automatically, so the entire visualization layer is reproducible and version-controlled alongside the rest of the system. Furthermore, Threema use Grafana internally for other monitoring purposes, so it is already a familiar tool for the team and can be easily integrated into existing workflows, this also influenced our decision in favor of Grafana.

4.5 Metrics Definition

As mentioned in Section 4.3.1, we decided that the bucket ranges for metrics will be hard-coded in the Threema client to prevent abuse of the system by malicious actors and allow users to gain insight into the bucket ranges because its open source. To maintain a common information base regarding the telemetry data collected by clients and the analytical processing performed on the backend, the system uses a centralized configuration file: `metrics-definition.json`. An example of this file can be found in the attachments (Appendix A.11). This file serves as the definitive schema for all collectable metrics, defining parameters such as metric IDs, bucket ranges and descriptions. A copy of this file is included in the client application bundle as well as in the metrics server, ensuring that both sides of the system operate with a consistent understanding of the current metrics being collected.

4.5.1 Updating the Schema

We have deliberately chosen an offline, manual update approach for this schema. Any changes to the `metrics-definition.json` file must be bundled within a formal client application update. Additionally, the metrics server must be updated as well. If a metric type is updated (e.g. bucket ranges are changed), it will receive a new metric ID. This design choice ensures that outdated clients will not recognize new metrics with changed

buckets, thus not reporting them until the client receives the update with the new metric ID.

While this approach introduces a slight increase in administrative overhead during the development cycle, it provides a privacy-preserving system with a transparent process behind it.

4.6 System Architecture

This section provides an overview of all components of the metrics system. Figure 4.2 displays a C4-style container diagram, including the 2 types of users the system serves:

- **App User:** The end user of the Threema app who can choose to opt-in to the metrics collection. He uses the Threema app with our built-in metrics package.
- **Threema Product Manager:** The product manager uses the configuration frontend as well as the metrics analysis tool to configure the metrics collection and analyze the collected data.

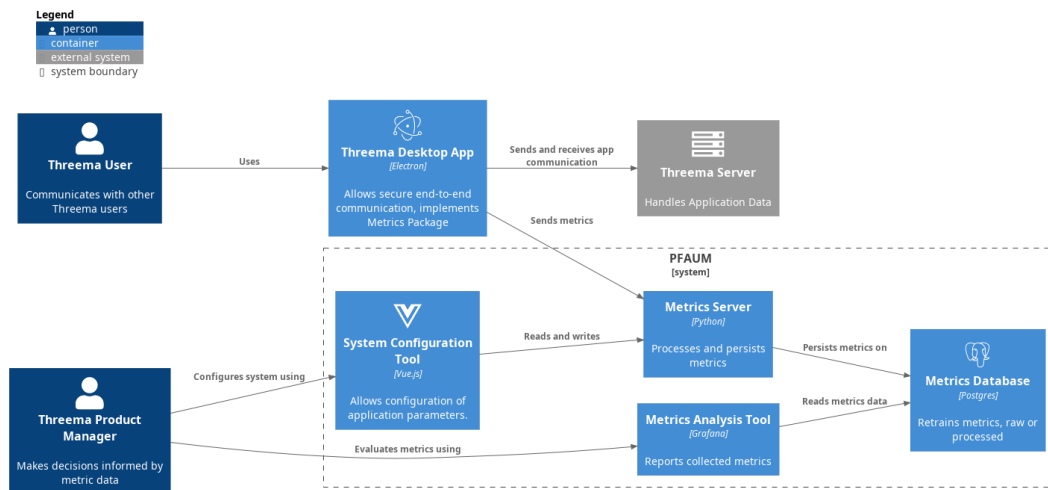


Figure 4.2: C4 Container Diagram displaying system components

The components of this system will be described in more detail in the following sections.

4.7 Threema Desktop App

This section will explain how the Threema desktop app has been changed and extended.

4.7.1 Package architecture

In this subsection we will provide an overview of the architecture of the developed JavaScript package. The package is written in TypeScript to ensure type safety and then compiled into JavaScript. Our package will consist of layers, each having their own functionality and responsibilities. Separation of responsibilities and having clear boundaries enables layers to be decoupled from another resulting in easier maintenance, better reusability and easier testing in addition to other benefits. [Mar17]

In Table 4.6 we have identified the layers our package will have and the functionality each layer provides.

Table 4.6: Overview of telemetry library layers

Layer	Functionality
Collection	Provides endpoints to submit telemetry directly and hooks into the HTML structure to listen for user inputs.
Data Access	Stores data that accumulates while the user is using the app and retrieves it on request.
Scheduling	Handles logic about conditions when telemetry can be submitted to the backend server.
Calculation	Performs calculations on existing data such as ratios between numbers or spans between times.
Privacy	Performs differential privacy algorithms and bucketing before sending data to a backend server.
Network	Connects to the backend server, serializes and sends telemetry data, receives backend server configuration, and selects metrics randomly.
Configuration	Provides configurable parameters for telemetry collection, differential privacy, transmission intervals, and other settings.

You can see a UML diagram of the layers and how they interact with each other in figure 4.3. Figure 4.4 shows how the package lives in the Threema desktop app and how it communicates with the backend server.

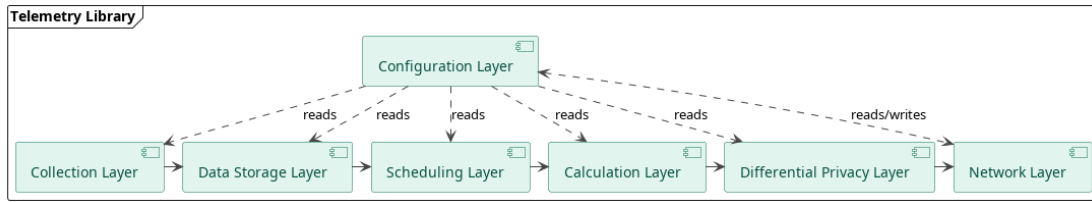


Figure 4.3: UML overview of package architecture

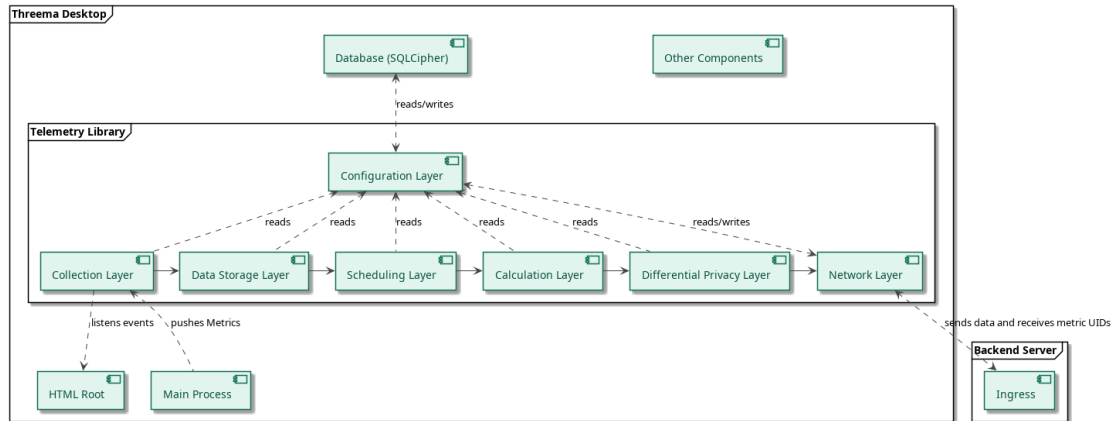


Figure 4.4: UML view of where the package lives in the Threema desktop app and how it communicates

4.7.1.1 Collection Layer

The Collection layer is responsible for collecting event data and measured data. There are two ways data can be collected. One way is by attaching an event listener to the HTML-Root of the electron app and listen for click-events. The eventlistener then searches for the closest element with an attribute *data-telemetry-key* and, if an element exists, gets the value for the attribute. For example, the copy-poll button has the attribute *data-telemetry-key=copy_poll*. If a click event is registered on that button, the collection layer pushes that event to the **data access layer**.

The second way is to call methods of the layer directly, like submitting the app start-time. When called, it pushes data to the **data access layer**.

4.7.1.2 Data Access Layer

This layer will provide API access to the Database. The layer will be accessed by the **collection layer** injecting data to the database and the scheduling layer, pulling data from the database. We have considered using the already existing database found in the Threema Desktop app or providing a parameter to inject the database connection

as a dependency into the layer. This would result in tight coupling by the package and the software it uses, meaning changes to the database structure affect both apps at the same time. In order to improve maintainability we will use a separate database for the storage of metrics and configuration.

We chose Dexie DB, a wrapper of the standard browser database indexedDB. Dexie allows for easier usage of the indexedDB in addition to implementing work-arounds for common bugs found in indexedDB. Dexie persists the data locally in a file, either where the electron process defines it or in the storage folder of a browser. Since our database will only do minor calculations such as grouping or joining, Dexie's lightweight nature fits our use case. In addition, using Dexie makes the package platform independent, not relying on nodeJS. Dexie allows easy inspection of stored data, improving trust and transparency to the users choosing to opt-in to collecting telemetry.

Figure 4.5 shows the entity relation diagram of the database. While Dexie does not enforce foreign key constraints, the logic is implemented in the code. Each app start creates one session and all events collected within that session are associated to it with the foreign key.

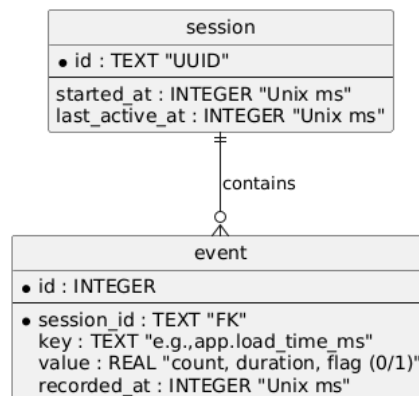


Figure 4.5: Database entity relation diagram

After the app has started and session has been created, it transmits all required data from the database to the **scheduling layer** to initiate the process of transmitting the telemetry data. Additionally, it will delete all data that is older than the current reporting window.

4.7.1.3 Scheduling Layer

This layer is responsible to check whether enough time has passed since the last time the app successfully transmitted metrics before passing the data from the data layer to the **calculation layer**. The required amount of time that must have passed can be set in the **configuration layer**. This layer can be extended if other time based conditions are needed to transmit telemetry data.

4.7.1.4 Calculation Layer

The calculation layer will perform calculations based on the metrics that will be sent. Some metrics require little calculation, for example "how many times has button X been pressed" is the sum of all button presses. Other metrics like return rate requires calculating the median time between sessions in minutes.

This layer also decides which metrics are calculated and sent to the next layer based on the metrics defined in the *metrics-definition.json* file. The file contains metric objects defined by their name, unique identifier and their buckets. For each name of the metrics, the layer defines a mapping to a function that calculates the value of the metric.

In conclusion, depending on what metrics are to be sent, this layer will receive the raw database data from the scheduling layer and perform necessary calculations before passing it to the [privacy layer](#).

4.7.1.5 Privacy Layer

This layer will receive the values from the calculation layer as numbers and perform privatizing logic on the data received.

In a first step, the values are put in their corresponding buckets as defined in the *metrics-definition.json* file according to this pattern:

$$(-\infty, b_0] \mid (b_0, b_1] \mid (b_1, b_2] \mid \cdots \mid (b_{n-1}, +\infty)$$

where $b = (b_0, b_1, \dots, b_{n-1})$ is the bucket with its index. In the second step, noise can be added to the bucket index. This can be disabled per metric since some metrics might not have buckets on a linear number scale, such as for example operating system names. Noise is added using the Laplace mechanism with a random number received from the cryptographically secure *crypto* library since the *Math* library implements predictable number generation. This added noise makes the transmitted bucket index shift by a certain probability. For example, with a sensitivity of 1 and an epsilon of 2, there is a 63.2% chance to stay in the same bucket and a 31.8% chance to jump one bucket to the left or right. We chose an epsilon of 2 since it is a good balance of additionally obscuring transmitted data and data accuracy. Finally, the metric, along with the bucketed index, is sent to the [network layer](#) for transmission.

4.7.1.6 Network Layer

This layer is responsible for sending the calculated metrics from the differential privacy layer to the backend server. Before the metric values can be sent, the layer must know how the defined weights of the metric types and how much entropy budget it has available. Then, depending on what the budget allows, metrics are selected based on the received weights. After each transmission attempt, the network layer records the timestamp in the [configuration layer](#), allowing other components, such as the [scheduling layer](#), to track when contact with the backend server was last made.

4.7.1.7 Configuration Layer

The configuration layer serves the purpose of holding configuration data that can be read and write from. It is designed as a singleton that lives in memory. Since part of the configuration needs to be persisted across multiple lifecycles of the app, the configuration layer receives an adapter that reads and writes to a database, in this case the SQLCipher database of the Threema desktop app. The decoupling of the data access layer and the configuration storage location avoids cyclic dependency and keeps the responsibility boundaries clean.

4.7.2 Opt-in

The opt-in flow is the part of the system that the user interacts with directly, so it has to integrate seamlessly into the existing Threema Desktop application. It consists of two elements: A consent dialog that is shown when the application starts (FR-C01) and a toggle in the privacy settings that lets the user change their decision at any time (FR-C03). The dialog uses the primary/secondary button layout that was selected together with Threema during the design phase (Section 3.5.2.2), and the settings toggle follows the corresponding mockup (Figure A.1), resulting in the implemented setting shown in Figure A.11.

The Threema Desktop frontend follows an atomic design approach, in which the user interface is composed from small, reusable building blocks organized into *atoms*, *molecules* and *partials*. Rather than building new components from scratch, we implemented both the consent dialog and the settings toggle by composing these existing building blocks. This made the opt-in very pleasant to incorporate: Because the dialog and the toggle are assembled from the same atoms and molecules as the rest of the application, they automatically match its look and feel and behave consistently. Additionally, the code should be easy to maintain as we closely followed the existing patterns and practices of the Threema Desktop codebase.

4.8 Metrics Server

The metrics server, contained in the backend, receives telemetry batches from the clients, stores them, derives the privacy parameters of the collected data and serves configuration to the clients and the administration frontend. It is implemented in Python using the FastAPI web framework, the SQLAlchemy object-relational mapper and communicates with a PostgreSQL database.

Internally, the server is organized into three layers. The router layer exposes the HTTP API and validates incoming requests, the service layer contains the privacy logic (entropy budget and k-anonymity computation), and the data access layer maps the domain entities to the database through SQLAlchemy.

4.8.1 Data Model

The persisted data is organized into the entities described in Table 4.7 and shown in Figure A.2. A batch groups the metrics submitted in a single transmission, and each metric references the metric type it measures. Configuration is stored as an append-only history: Every change to a config key creates a new config change, which is what allows the configuration timeline to be reconstructed in the dashboards.

Table 4.7: Entities of the metrics server.

Entity	Purpose
MetricType	A type of metric (for example the application load time), including its name, description, static bucket ranges and recording weight.
Metric	A single bucketed value of a metric type, belonging to a batch.
Batch	The set of metrics submitted together in one transmission, without any persistent identifier.
SendRequest	A record that a client requested a submission window, used to count opt-in users for the entropy budget.
ConfigKey	A configurable system parameter, such as the target k-anonymity or the recalculation interval.
ConfigChange	A history entry recording a new value for a config key.
PrivacyParameter	A computed privacy value (for example the measured k-anonymity) stored with a timestamp.

4.8.2 The API

The server exposes a REST API, summarized in Table 4.8. The endpoints fall into two groups: those consumed by the Threema client when submitting telemetry, and those consumed by the administration frontend to configure the system and read back its state. In addition, FastAPI automatically generates an interactive OpenAPI specification, which documents every endpoint and its schema without manual effort (Appendix A.10).

Table 4.8: Main endpoints of the metrics server.

Method	Endpoint	Purpose
Client		
POST	<code>/metrics/send-request</code>	Request a submission window returns the entropy budget and the metric weights.
POST	<code>/metrics/batch</code>	Submit a batch of metrics and trigger a privacy recalculation.
Administration		
GET	<code>/metrictypes</code>	List the metric types and their recording weights.
PUT	<code>/metrictypes/{id}</code>	Update the recording weight of a metric type.
GET	<code>/config-keys/with-values</code>	List the configuration keys with their current values.
PUT	<code>/config-keys/{id}/value</code>	Set a new value for a configuration key.

4.8.3 Telemetry Submission Flow

A client does not simply push its metrics to the server. It calls `/metrics/send-request` first, which records the request and returns the current entropy budget together with the metric weights. The entropy budget is computed based on various settings such as the selected calculation mode and the target k-anonymity, as well as the number of send requests received in the current interval. Based on this budget and the weights, the client selects which metrics it may report and then submits them through `/metrics/batch`. The complete flow, from local metric collection through reporting to the visualization, is shown in Section 4.12.

4.8.4 Privacy Parameter Computation

Whenever a batch is submitted, the server schedules an asynchronous recalculation of the privacy parameters. To avoid excessive computation, the recalculation only runs if the configured interval has elapsed since the last run, and a lock ensures that at most one recalculation runs at a time.

The recalculation computes the k-anonymity of the collected data, both over the full set of metrics and over subsets of different sizes. Each result is written to the database as a *privacy parameter* with a timestamp.

4.8.5 Initialization and Configuration

On startup, the server initializes the database if it is empty. The available metric types, including their static bucket ranges, are loaded from a `metrics-definition.json` file. The default configuration keys, the target k-anonymity and the recalculation interval, are seeded with initial values. Finally, the server installs a small Postgres function that maps a bucket index to a label, which is used by the Grafana dashboards (Section 4.4.4).

4.9 Multi-device Support

Threema may want to collect data from multiple devices to get insight on how different apps are used. This poses a challenge since some non-device-specific data might get reported multiple times for the same user (e.g. active contacts). As we have established in our concept, we are protecting individuals from being uniquely identified within the dataset by limiting the amount of information transmitted to the backend server. The amount of information that is transmitted is a function of how many individuals opted in to transmitting telemetry. Using multiple devices might now change that in two aspects:

4.9.1 Platform-specific Metric as Additional Entropy

If Threema chooses to evaluate some metrics only certain platforms and such a metric is transmitted by a user, we would learn about their platform usage without accounting for that additional information in the cost of entropy. Therefore, we would have to always send what platform the user is on, which would eliminate the problem. Another option is maintaining separate systems and user pools per platform, potentially reducing the amount of users opted-in which in turn reduces the entropy budget.

4.9.2 Transmitting Additional Information

If a user operates Threema on multiple devices, they may transmit more data within a single reporting window than the entropy budget permits. Consider a budget that allows four metrics: the first platform transmits its four allocated metrics, while the second platform independently selects and transmits its own set of four metrics. Since the two platforms select metrics independently, their selections may overlap only partially, resulting in a combined total that exceeds the budget.

4.9.3 Solution

Having looked at these two problems, we decided to implement a mechanism such that each user can only transmit telemetry once per reporting period across all platforms. This is realized by syncing the last time telemetry was transmitted in the settings which syncs across all platforms per default. As a new issue, this would introduce a potential race condition between platforms if the devices were to transmit telemetry at the exact same time. If Threema implemented a Zero-Knowledge Proof to block a single user from

transmitting telemetry multiple times as described in Section 2.3, this problem would be alleviated.

Additionally, Threema would have to transmit a metric describing the platform the user uses if they were to use platform-specific metrics. This metric would have to be always transmitted, together with the rest of the randomly selected metrics.

4.10 Administration Dashboards

The visualization part of the administration tooling is realized with Grafana (Section 4.4.4). All dashboards read directly from the metrics PostgreSQL database through a provisioned Postgres data source, and every panel is backed by an SQL query. The data source and the dashboards are checked into the repository as configuration, a single data source definition and one JSON file per dashboard, and are loaded automatically when Grafana starts. This means the whole visualization layer can be recreated from scratch without any manual setup.

The prototype provides five dashboards, each covering one aspect of the collected data, as listed in Table 4.9.

Table 4.9: Grafana dashboards of the administration tooling.

Dashboard	Description	Req.	Figure
Metrics Dashboard	Shows the distribution of each metric across its buckets and how the number of submitted metrics develops over time.	FR-A4	A.17
Privacy Parameters	Visualizes various k-anonymity calculations of the collected data	FR-A8	A.18
Config Keys Timeline	Shows the history of configuration changes, such as the target k-anonymity value and the recalculation interval.	FR-A5	A.19
SendRequests & Batches	Monitors the volume of received send requests and metric batches over time.	FR-A6	A.20
Metrics Correlation Analysis	Visualizes the correlation between different metrics and how it develops over time.	FR-A7	A.21

4.11 Configuration Frontend

The configuration frontend is the administrative interface through which the privacy and collection parameters of the system are managed. Together with the Admin Dashboard (Section 4.10), they form the administration tooling and are internal tools that are not exposed to end users.

The frontend is implemented as a single-page application using the Vue 3 framework with TypeScript, built and bundled with Vite. It holds no data of its own and no business logic, every value it shows is loaded from and every change it makes is written back to the metrics server through its administration REST API (Section 4.8). The address of the metrics server is injected at build time through the `VITE_API_BASE_URL` environment variable, so the same code base can target different backends without modification.

4.11.1 Structure

The application is organized around a small set of views connected by the client-side router, summarized in Table 4.10. A persistent navigation bar links the two functional views, the metric weight configuration and the privacy budget configuration, which cover the three configuration-related administration requirements [FR-A1](#), [FR-A2](#) and [FR-A3](#). Screenshots of all screens are collected in Appendix [A.8](#).

Table 4.10: Views of the configuration frontend.

View	Purpose	Req.	Figure
Home	Landing page that links to the two functional views.	–	A.12
Metric weights	Lists the metric types and lets the administrator adjust their recording weight.	FR-A2	A.13
Privacy budget config	Edits the privacy parameters, such as the target k-anonymity and the recalculation interval.	FR-A1 , FR-A3	A.14 , A.15 , A.16

4.11.2 Metric Weight Configuration

The metric weights view (Figure [A.13](#)) loads the metric types from the server (`GET /metrictypes`) and renders one card per metric type. Each card carries a slider and a numeric input bound to the same draft value, so the weight can be set either coarsely by dragging or precisely by typing. The weight ranges from 0 to 5 and is displayed as a

percentage relative to the neutral weight of 1. To make that neutral value easy to hit, the control snaps to 1 whenever the draft lands within a small tolerance around it.

The view tracks which cards differ from their saved value and only enables the save action when there are pending changes. Saving sends one update per changed metric type (`PUT /metrictypes/{id}`) and reports the outcome through a toast notification, so a partial failure, where some weights are saved and others are not, is made visible rather than hidden. The recording weight influences the probability that a client reports the corresponding metric within its entropy budget ([FR-C13](#)).

4.11.3 Privacy Budget Configuration

The privacy budget view edits the configuration keys that the server stores as an append-only history (Section [4.8](#)). It loads the keys with their current values (`GET /config-keys/with-values`) and, on save, writes a new value for each changed key (`PUT /config-keys/{id}/value`), which on the server side creates a new configuration change. This is the same history that the configuration timeline dashboard later reconstructs. The keys that the view exposes are listed in Table [4.11](#).

Table 4.11: Configuration keys exposed by the configuration frontend.

Key	Meaning
Budget model	Selects how the entropy budget is derived: a simple overall estimate (Figure A.14), an average-subset estimate (Figure A.15), or a worst-case minimum-subset estimate (Figure A.16).
Target k	The target k-anonymity that the selected budget model aims for.
Subset size	The number of metric dimensions considered when reasoning about subset k-anonymity.
Minimum-k assumptions	The assumed probability and count of the rarest bucket combination, used only by the minimum-subset model.
Budget cap	A hard upper bound on the entropy budget returned to clients.
Recalculation interval	How often the cached privacy parameters are recomputed.

The view adapts to the selected budget model: the controls that are only meaningful for a given model, such as the subset size or the minimum-k assumptions, are shown only when that model is active, which keeps the interface focused on the parameters that actually take effect. Each parameter is given a control suited to its meaning rather than

a generic text field, the target k-anonymity and the budget cap use bounded sliders, the subset size a stepper, and the recalculation interval is entered in hours and minutes and converted to the minutes the server expects ([FR-A3](#)). Any configuration key that does not yet have a dedicated control falls back to a raw text field, so newly introduced keys remain editable without a code change. As in the metric weights view, the save action is enabled only when there are unsaved changes and its result is surfaced through a notification.

4.11.4 Styling

To keep the configuration frontend visually consistent with the rest of the product, its styling is derived from the Threema stylesheet rather than designed from scratch. The color palette, in particular the dark surfaces and the signature green accent used for active controls and primary actions, as well as the typography are taken from Threema's design language and reused throughout the views. This keeps the administration tooling recognizable as part of Threema and avoids introducing a separate, unrelated visual identity for an internal tool.

4.12 Application Flow

The complete flow of the system, from local metric collection on the client, through the submission process, to the visualization in Grafana, is shown in [Figure 4.6](#).

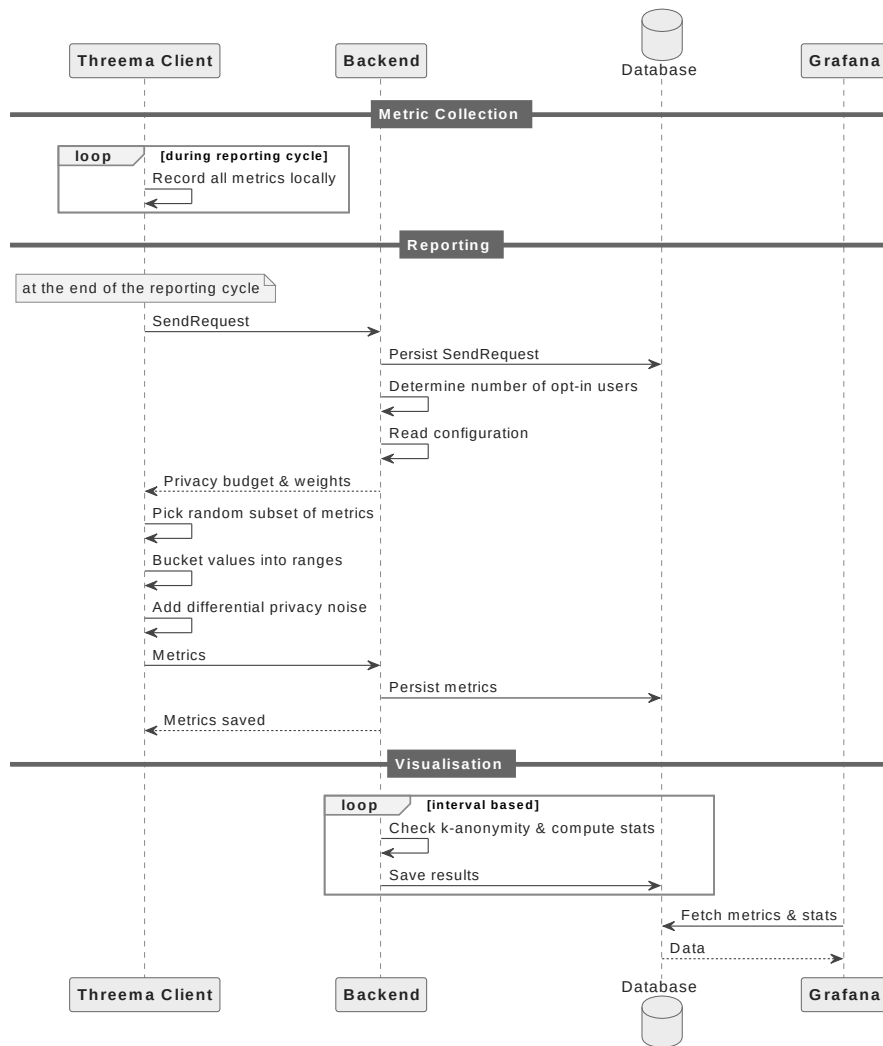


Figure 4.6: Sequence of a telemetry submission, from collection on the client to visualization.

Chapter 5

Evaluation and Results

This chapter evaluates the implemented prototype and presents the results obtained from it. The evaluation follows the goal stated in Section 1.3: To assess the prototype in terms of privacy, performance, usability and its suitability for a potential integration into the Threema app.

5.1 Testing Results

This section presents the results obtained from the evaluation methods described in Chapter 3.6. The objective is to assess whether the proposed telemetry system achieves its intended privacy guarantees and to identify any practical limitations of the design.

The prototype is a proof of concept, so the evaluation focuses on validating the system design and its proposed benefits and weaknesses rather than production-readiness.

5.1.1 Mock Data Evaluation

As an initial validation step, the system was tested using mock data without random sampling enabled. The results of these tests (Appendix A.5.1) confirmed that the system behaved as expected and that the implemented anonymity calculations produced the anticipated results. After establishing the correctness of the baseline system, additional experiments were conducted with random sampling enabled. As discussed in Section 3.3.2, the resulting datasets are not directly compatible with traditional k-anonymity calculations, so the proposed subset method was used instead. Synthetic datasets emulating telemetry data from large numbers of Threema users were generated and evaluated using multiple value generators and analysis tools. The observed anonymity values were then compared against the predictions made by the formulas developed in this thesis (Appendix A.5.3). These experiments provided valuable insight into the behavior of the system and enabled the refinement of the proposed anonymity estimation formulas. In particular, the two corrective terms in the formula for the expected minimum k-anonymity value (Equation 3.6) were derived from observations made during extensive testing under different conditions. Overall, the results demonstrated that the formulas

provide useful estimates of anonymity values and that random sampling further reduces the re-identification risk for users. The experiments therefore support the claim that the system achieves the intended privacy goals.

5.1.2 Unit Test Results

Unit tests were used to validate the behavior of the client-side implementation. The tests verified that the privacy-preserving mechanisms were implemented according to their specification and that the client correctly enforced the intended constraints before transmitting telemetry data. Since the Threema client is responsible for enforcing the privacy guarantees and can be publicly audited, successful unit test execution increases confidence in the correctness of the implementation. The results showed that the tested components behaved as expected and correctly implemented the privacy-related functionality described in this thesis.

5.1.3 Network Test Results

The network tests investigated whether information about transmitted telemetry data could be inferred from packet sizes despite the use of TLS encryption. The experiments showed that packet sizes varied depending on the number of transmitted metrics. Figure 5.1 illustrates this behavior by comparing transmissions containing one metric and six metrics respectively. The observed packet sizes differed significantly, measuring approximately 370 bytes and 831 bytes. This behavior was reproducible and predictable, indicating that packet size could potentially be used as a side channel. An observer capable of monitoring network traffic could use this information to infer details about transmitted telemetry data and, under certain circumstances, reduce the anonymity of users.

320	1.058228620	152.96.163.194	152.96.10.31	TLSv1.3	327 Client Hello (SNI=srbsci-31.ost.ch)
321	1.064309495	152.96.10.31	152.96.163.194	TCP	66 443 → 51558 [ACK] Seq=1 Ack=262 Win=
322	1.066351575	152.96.10.31	152.96.163.194	TLSv1.3	2606 Server Hello, Change Cipher Spec, Ap
323	1.066367054	152.96.163.194	152.96.10.31	TCP	66 51558 → 443 [ACK] Seq=262 Ack=2541 W
324	1.067987930	152.96.163.194	152.96.10.31	TLSv1.3	130 Change Cipher Spec, Application Data
325	1.068091556	152.96.163.194	152.96.10.31	TLSv1.3	370 Application Data, Application Data
326	1.071158759	152.96.10.31	152.96.163.194	TLSv1.3	127 Application Data
327	1.071158939	152.96.10.31	152.96.163.194	TLSv1.3	101 Application Data
328	1.071159060	152.96.10.31	152.96.163.194	TLSv1.3	97 Application Data
329	1.071192312	152.96.163.194	152.96.10.31	TCP	66 51558 → 443 [ACK] Seq=630 Ack=2668 W

(a) Wireshark packet inspection with one metric sent (packet 325)

[6pt]

751	1.685559930	152.96.163.194	152.96.10.31	TLSv1.3	327 Client Hello (SNI=srbsci-31.ost.ch)
752	1.688966704	152.96.10.31	152.96.163.194	TCP	66 443 → 59872 [ACK] Seq=1 Ack=262 Win=
753	1.689019032	152.96.10.31	152.96.163.194	TLSv1.3	2604 Server Hello, Change Cipher Spec, Ap
754	1.689030444	152.96.163.194	152.96.10.31	TCP	66 59872 → 443 [ACK] Seq=262 Ack=2539 W
755	1.690736781	152.96.163.194	152.96.10.31	TLSv1.3	130 Change Cipher Spec, Application Data
756	1.690842290	152.96.163.194	152.96.10.31	TLSv1.3	831 Application Data, Application Data
757	1.694224888	152.96.10.31	152.96.163.194	TLSv1.3	127 Application Data
758	1.694225048	152.96.10.31	152.96.163.194	TLSv1.3	101 Application Data
759	1.694225169	152.96.10.31	152.96.163.194	TLSv1.3	97 Application Data
760	1.694261677	152.96.163.194	152.96.10.31	TCP	66 59872 → 443 [ACK] Seq=1091 Ack=2666 W

(b) Wireshark packet inspection with six metrics sent (packet 756)

Figure 5.1: Difference in network packet size when transmitting one and six metrics respectively (370 vs. 831 bytes)

To mitigate this issue, packet padding was introduced. Dummy metrics were added to telemetry submissions such that all packets were transmitted with a constant size. The backend server discarded these dummy entries during processing. Subsequent testing confirmed that packet sizes remained constant regardless of the number of transmitted metrics, effectively eliminating this side channel.

5.1.4 Discussion

While the testing results generally support the claims made in this thesis, it is important to acknowledge that the evaluation was conducted under controlled conditions using synthetic data and a prototype implementation. Real-world usage may reveal additional challenges and limitations that were not captured in this testing phase.

5.1.5 Conclusion

The evaluation confirmed that the telemetry system achieves the objectives defined in the testing concept. The system behaved predictably throughout the experiments, the proposed anonymity models were validated, and identified privacy risks were successfully mitigated. These results provide confidence that the overall design is sound and that the benefits and limitations discussed throughout this thesis accurately reflect the behavior of the implemented system.

5.2 Requirements Fulfillment

Table 5.1 summarizes to what degree the prototype fulfills the functional requirements from Section 4.1.

Table 5.1: Fulfillment of the functional requirements by the prototype.

ID	Status	Evidence / Notes	Priority
Threema App Client			
FR-C01	✓	Opt-in popup shown on application startup.	Must
FR-C02	✓	Decision gets persisted through the configuration layer.	Must
FR-C03	✓	The toggle available in the application settings.	Must
FR-C04	✓	The collection of metrics only starts after opting in.	Must

continued on next page

ID	Status	Evidence / Notes	Priority
FR-C05	✓	Event listener reacts on elements with the attribute ‘data-telemetry-key‘.	Must
FR-C06	✓	Local storage using DexieDB until submission, then the data is cleared.	Must
FR-C07	✓	Bucketing is performed in the privacy layer.	Must
FR-C08	✓	Laplace noise is applied to the bucket index.	Must
FR-C09	✓	Noise can be disabled per metric.	Must
FR-C10	✓	Identifier-free batches with padding (Sec. 2.3).	Must
FR-C11	✓	The application supports a configurable reporting window.	Must
FR-C12	✓	The metrics only get sent if enough users have opted in.	Must
FR-C13	✓	Metric selection by entropy budget and server weights.	Should
FR-C14	✓	Parameters configurable via the configuration layer.	Should
FR-C15	Partially	Local data can be inspected using external tools but not natively supported in the application.	Could
FR-C16	✓	The setting gets persisted across platforms.	Could
Server			
FR-S1	✓	REST endpoint over TLS.	Must
FR-S2	✓	The server stores the received telemetry in a Postgres database.	Must
FR-S3	✓	Client IP addresses are not persisted.	Must
FR-S4	✓	IP-based rate limiting is implemented via Traefik.	Could
Administration Tooling			
FR-A1	✓	Target k-anonymity is configurable in the admin frontend.	Must

continued on next page

ID	Status	Evidence / Notes	Priority
FR-A2	✓	Per-metric weight is configurable in the admin frontend.	Must
FR-A3	✓	Recalculation interval is configurable in the admin frontend.	Must
FR-A4	✓	Dashboard in Grafana for metric distributions and their development over time.	Must
FR-A5	✓	Dashboard in Grafana showing the history of configuration changes.	Should
FR-A6	✓	Dashboard in Grafana monitoring received request and batch volume.	Should
FR-A7	✓	Cross-metric correlation visualization in Grafana dashboard.	Could
FR-A8	✓	Dashboard in Grafana visualizing the different k-anonymity calculations of the collected data and its history.	Could

Overall, the prototype fulfills every must-have and should-have requirement. Only one of the two could-have requirements, inspecting the locally stored data (FR-C15), is only partially fulfilled, as it is not essential for demonstrating the concept.

5.3 Privacy Evaluation

Privacy is the central quality of this system, so we evaluate it in the most depth. We assess three aspects: How well the collected metrics protect users against re-identification, how accurately our k-anonymity formulas predict this protection, and whether the data transmission leaks information at the network level.

5.3.1 Re-identification Risk

Using the formulas from Section 3.3.3, we compared the predicted average and minimum k-anonymity values against the values measured on mock data (Appendix A.5.1 and Appendix A.5.3). The results show that for our simulated scenarios, the formulas provide an accurate prediction of k-anonymity values. Following this, we verified that these formulas are applied properly in the implementation, and that the system sets a budget that is consistent with the target k-anonymity value. Having established that the system correctly enforces the k-anonymity targets, we can be confident that the collected metrics provide the expected level of protection against re-identification.

5.3.2 Network-Level Information Leakage

Even with encrypted transmission, the size of a metrics batch can leak which metrics were sent (Section 2.3). We demonstrated this with packet captures (Figure 5.1) and showed that padding all batches to a constant size removes the leak.

5.4 Limitations

While the prototype successfully demonstrates the feasibility of a privacy-friendly metrics system, it is important to be aware of its limitations:

- **Estimate, not guarantee:** True k-anonymity of a metric combination cannot be guaranteed before the real dataset exists. The system has to rely on estimates based on assumptions and requires ongoing monitoring and adjustments.
- **Static buckets:** Choosing static bucket ranges protects against an evil server but prevents post-launch tuning, resulting in a less flexible system.
- **Authentication fallback:** A system to reliably prevent misuse (fake data submission, DoS attacks) without user identification requires complicated schemes such as zero-knowledge proofs, which requires a significant development effort.

While this system aims for "Zero-trust", it still relies on some level of trust in the server operator in several aspects like the honesty of k-anonymity estimates and its corresponding budget, the discarding of IP addresses and the overall security of the server infrastructure. While this trust can be somewhat mitigated by independent audits, open communication with the user base and transparency about the collected data will be required to maintain user trust in the system.

5.5 Discussion

At the beginning of this thesis, k-anonymity was chosen as one of the central privacy objectives because it provides an intuitive and widely established framework for measuring re-identification risk. It also aligned well with our understanding of the project's goal: Preventing users from being singled out based on combinations of telemetry attributes.

During the design and implementation process, however, several challenges emerged: Classical k-anonymity is defined on an existing dataset, whereas our system must evaluate privacy properties before data is collected. The need to collect more data also led to random sampling, which introduces nullable values. This effectively renders traditional k-anonymity values inapplicable. This required the introduction of entropy-based approximations, relying on assumptions about metric combinations, bucket distributions and attacker knowledge. As a result, much of our work focused on developing a framework for estimating k-anonymity in a setting it was never meant for.

The resulting framework proved usable and enabled the implementation of a privacy-aware telemetry system which puts re-identification into empirical numbers. However, the complexity and assumptions involved suggest that k-anonymity should perhaps have been treated as one possible evaluation criteria rather than the primary design objective. Additionally, many of the developed mechanisms, such as random sampling or differential privacy contribute to privacy protection while not influencing the k-anonymity in a meaningful way.

In retrospect, being more open to a wider range of privacy-preserving techniques might have helped with constructing a more general model with fewer assumptions. K-anonymity nevertheless provided a useful conceptual foundation, and helped guide design decisions throughout the project, even if the final solution ultimately relied on several extensions and approximations beyond the traditional definition of k-anonymity.

Chapter 6

Conclusion

This chapter summarizes the work done in this bachelor thesis, reflects on the results obtained and discusses potential future work.

6.1 Summary

This thesis set out to give Threema a way to learn how its app is used without compromising the privacy that defines it. We designed a privacy-friendly, opt-in telemetry concept built on entropy as an estimate for k-anonymity: An entropy budget defines how much a set of metrics can reveal, while bucketing, differential privacy and random sampling keep individual users protected.

We realized this concept as a prototype consisting of the five parts described in Table 6.1.

Table 6.1: Parts of the prototype.

Part	Description
Metrics package	A metrics collection package embedded in the Threema Desktop client.
Opt-in flow	A client-side flow that asks the user for consent before any data is collected.
Backend	A server that receives and aggregates the submitted batches.
Administration tool	A tool that lets an operator configure the privacy parameters and the metric weights.
Grafana	A set of dashboards that visualize the collected data.

Together they complete the full scope, from opting in, to collecting an event on the client, to viewing the results.

While our entropy-based approach to estimating k-anonymity presented some challenges throughout the project, we consider the approach feasible. It is a valid basis for Threema to decide on real-world integration, provided the privacy guarantees are re-validated once actual opt-in data becomes available.

6.2 Future Work

The prototype validates the concept, but several directions remain open for a production-grade system and for closing the limitations discussed in Section 5.4. These directions are summarized in Table 6.2.

Table 6.2: Possible directions for future work.

Direction	Description
Anonymizing transport	The server could still observe client IP addresses, which remain a residual linkability risk if persisted. Routing transmissions through an anonymizing network such as Tor would remove this risk at the cost of additional complexity.
Scalable time-series storage	Adopting a time-series database such as TimescaleDB could improve aggregation and historical queries as the collected data grows.
Richer dashboards	The administration dashboards could be extended with additional views and metrics to give the product team deeper insight into the collected data.
Real-world testing	Deploying the system with real Threema users would validate the privacy, performance and usability assumptions in practice, and in particular allow the k-anonymity guarantees to be checked against an actual dataset.
Anonymous authentication	Replacing the IP-based rate limiting with a zero-knowledge-proof scheme (Section 2.3) would let the metrics server authenticate genuine clients without revealing their identity. Additionally, rate limiting could be done per user.
Multi-platform support	Extending the metrics package beyond Threema Desktop to the mobile clients would broaden the data basis significantly. However, this would require additional platform-specific functionality beyond a straightforward port (see Section 4.9).

6.3 Concluding Remarks

Threema’s existing privacy guarantees are technical and verifiable: end-to-end encryption, no required identifiers, open-source clients. This thesis tries to extend that principle to telemetry. Anonymization is enforced at client-side for collection rather than a post-processing step.

The system collects metrics across engagement, retention, communication, feature adoption, performance, and social graph categories. Re-identification risk is controlled through bucketing, Laplace noise, identifier-free transmission, and a dynamic entropy budget that gates submissions until a sufficient user population has opted in.

The main open question is one the system cannot resolve on its own: The entropy-based k-anonymity estimates rely on various assumptions that hold reasonably well in simulation but require validation against real data. A production deployment would need to monitor actual k-anonymity values and adjust parameters accordingly.

However, even if these entropy-based estimates turn out to be too optimistic, the collected data does not become usable for re-identification. The estimate only governs how much data is collected, not whether an individual can be singled out from it. The latter is protected by the combination of differential privacy, bucketing and random sampling, which together make it very unlikely that the actual values, or the quasi-identifiers an attacker might already know about a person, can be located in the dataset. And even if a record could be linked to a person, the differential privacy noise means the attacker can never be sure whether the reported values are genuine or were altered, leaving them without any reliable knowledge to act on. In this sense, the prototype demonstrates that a privacy-focused application can collect meaningful usage data without compromising the end-user privacy.

Bibliography

- [AAK⁺24] Esraa Alhenawi, Shatha Awawdeh, Ruba Abu Khurma, Maribel García-Arenas, Pedro A. Castillo, and Amjad Hudaib. Choosing a suitable requirement prioritization method: A survey, 2024.
- [Air26] Airship. Performance analytics dashboard and look definitions, 2026. Accessed: 2026-04-24.
- [Amp24] Amplitude, Inc. Churn rate formula: How to calculate and analyze churn, 2024. Accessed: 2026-04-24.
- [Amp25] Amplitude, Inc. You’re measuring daily active users wrong, 2025. Accessed: 2026-04-24.
- [Amp26a] Amplitude, Inc. Amplitude documentation, 2026. Accessed: 2026-04-24.
- [Amp26b] Amplitude, Inc. Analyze the adoption of a feature, 2026. Accessed: 2026-04-24.
- [Amp26c] Amplitude, Inc. How amplitude computes conversions through funnels, 2026. Accessed: 2026-04-24.
- [Amp26d] Amplitude, Inc. Interpret your lifecycle chart, 2026. Accessed: 2026-04-24.
- [Amp26e] Amplitude, Inc. Retention rate: Complete definition & calculation guide, 2026. Accessed: 2026-04-24.
- [Amp26f] Amplitude, Inc. User sessions: Track engagement frequency and duration, 2026. Accessed: 2026-04-24.
- [Apt26] Aptabase. Aptabase • Analytics for Mobile, Desktop and Web apps, April 2026. [Online; accessed 8. Jun. 2026].
- [Bez07] Michele Bezzi. An entropy based method for measuring anonymity. In *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops - SecureComm 2007*, pages 28–32, 2007.
- [CAM⁺23] Sandeep Chowdhary, Elsa Andres, Adriana Manna, Luka Blagojević, Leonardo Di Gaetano, and Gerardo Iñiguez. Temporal patterns of reciprocity in communication networks. *EPJ Data Science*, 12:7, 2023.

- [CEK14] Margareta Ciglic, Johann Eder, and Christian Koncilia. k-anonymity of microdata with null values. volume 8644, pages 328–342, 09 2014.
- [Cle26] CleverTap. 10 push notification metrics you need to track: Ctr, open rate & more, 2026. Accessed: 2026-04-24.
- [Com24] Commerce Gurus. Converting opt-in pages & popups, 2024. Accessed: 2026-03-17.
- [Comnda] Commerce Gurus. Ecommerce popup design, n.d. Accessed: 2026-03-17.
- [Comndb] Commerce Gurus. Exit intent popups, n.d. Accessed: 2026-03-17.
- [Did24] Didomi. Cmp - what is the difference between consent rate and opt-in rate, 2024. Accessed: 2026-03-17.
- [DMW20] Alexandre De Masi and Katarzyna Wac. Towards accurate models for predicting smartphone applications’ QoE with data from a living lab study. *Quality and User Experience*, 5(1):10, 2020.
- [Ful26] Fullstory. Fullstory help center and developer documentation, 2026. Accessed: 2026-04-24.
- [GG22] Aayush Gupta and Kobi Gurkan. PLUME: An ECDSA nullifier scheme for unique pseudonymity within zero knowledge proofs. *Cryptology ePrint Archive*, Paper 2022/1255, 2022.
- [GKM25] Johannes Gehrke, Daniel Kifer, and Ashwin Machanavajjhala. ℓ -diversity. In Sushil Jajodia, Pierangela Samarati, and Moti Yung, editors, *Encyclopedia of Cryptography, Security and Privacy*. Springer, Cham, 2025.
- [GMR85] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC ’85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery.
- [Goo25a] Google LLC. Firebase performance monitoring: Automatic traces, 2025. Accessed: 2026-04-24.
- [Goo25b] Google LLC. Understand crash-free metrics, 2025. Accessed: 2026-04-24.
- [Goo26a] Google LLC. [ga4] automatically collected events, 2026. Accessed: 2026-04-24.
- [Goo26b] Google LLC. [ga4] default channel group, 2026. Accessed: 2026-04-24.
- [Goo26c] Google LLC. [ga4] enhanced measurement events, 2026. Accessed: 2026-04-24.

- [Goo26d] Google LLC. [ga4] user stickiness, 2026. Accessed: 2026-04-24.
- [LH08] Jure Leskovec and Eric Horvitz. Planetary-scale views on a large instant-messaging network. In *Proceedings of the 17th International Conference on World Wide Web (WWW '08)*, pages 915–924. ACM, 2008.
- [Li25] Ninghui Li. T-closeness. In Sushil Jajodia, Pierangela Samarati, and Moti Yung, editors, *Encyclopedia of Cryptography, Security and Privacy*. Springer, Cham, 2025.
- [LKF07] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Den-sification and shrinking diameters. *ACM Transactions on Knowledge Discov-ery from Data*, 1(1), 2007.
- [LLYTD12] Janette Lehmann, Mounia Lalmas, Elad Yom-Tov, and Georges Dupret. Models of user engagement. In *User Modeling, Adaptation, and Personal-ization (UMAP 2012)*, Lecture Notes in Computer Science, pages 164–175. Springer, 2012.
- [LWPS18] Zijian Lew, Joseph B. Walther, Augustine Pang, and Wonsun Shin. Inter-activity in online chat: Conversational contingency and response latency in computer-mediated communication. *Journal of Computer-Mediated Commu-nication*, 23(4):201–221, 2018.
- [Mar17] Robert C. Martin. *Clean Architecture: A Craftsman’s Guide to Software Structure and Design*. Pearson, 2017.
- [McC07] Dave McClure. Startup metrics for pirates: Aarr, 2007. Accessed: 2026-03-09.
- [Mix26a] Mixpanel, Inc. Aarr! mixpanel for pirates, 2026. Accessed: 2026-04-24.
- [Mix26b] Mixpanel, Inc. Mixpanel documentation, 2026. Accessed: 2026-04-24.
- [MLCM13] Giovanna Miritello, Rubén Lara, Manuel Cebrian, and Esteban Moro. Lim-ited communication capacity unveils strategies for human interaction. *Scien-tific Reports*, 3:1950, 2013.
- [MMG⁺07] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social net-works. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC '07)*, pages 29–42. ACM, 2007.
- [Moz26] Mozilla. Glean - Glean SDKs - Cross-platform Telemetry Libraries, June 2026. [Online; accessed 8. Jun. 2026].
- [NA21] Joseph P. Near and Chiké Abuah. *Programming Differential Privacy*, vol-ume 1. 2021.

- [One25] OneSignal. Push notification message reports, 2025. Accessed: 2026-04-24.
- [OT22] Artyom Novopolsky Olga Trushkova. Att opt-in rates in ios 14.5+ by country and app category after 1 year, 2022. Accessed: 2026-03-05.
- [PdOKO14] Martin Pielot, Rodrigo de Oliveira, Haewoon Kwak, and Nuria Oliver. Didn't you see my message? predicting attentiveness to mobile instant messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*, pages 3319–3328. ACM, 2014.
- [Pos26] PostHog. Documentation - PostHog, June 2026. [Online; accessed 7. Jun. 2026].
- [SPHS22] Anika Seufert, Fabian Poignée, Tobias Hofffeld, and Michael Seufert. Pandemic in the digital age: Analyzing WhatsApp communication behavior before, during, and after the COVID-19 lockdown. *Humanities and Social Sciences Communications*, 9:154, 2022.
- [SSHTG15] Michael Seufert, Anika Schwind, Tobias Hofffeld, and Phuoc Tran-Gia. Analysis of group-based communication in WhatsApp. In *Mobile Networks and Management (MONAMI 2015)*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 225–238. Springer, 2015.
- [Tel26] TelemetryDeck GmbH. Quick Start Guide, February 2026.
- [Tes23] Tesk. Opt-in telemetry and asking users for feedback may not work in practice, 07 2023. Accessed: 2026-03-17.
- [Thr26a] Threema. Privacy policy for the use of threema (extended) — crash reports, 2026. Accessed: 2026-06-08.
- [Thr26b] Threema. Threema - the secure messenger, 2026. Accessed: 2026-02-23.
- [TKLA23] Pasan Tennakoon, Supipi Karunathilaka, Rishikeshan Lavakumar, and Janaka Alawatugoda. Anonymous and distributed authentication for peer-to-peer networks. *Journal of Computer Science*, 19:1–10, 01 2023.

List of Tables

1.1	Key elements of Threema.	1
3.1	Constraints of the System	8
3.2	Requirements for metrics	9
3.3	Assumptions made to calculate entropy budget.	12
4.1	MoSCoW prioritization legend	24
4.2	Functional requirements of the Threema client	26
4.3	Functional requirements of the server	27
4.4	Functional requirements of the administration tooling	28
4.5	Comparison of existing telemetry tools against requirements	29
4.6	Overview of telemetry library layers	36
4.7	Entities of the metrics server.	41
4.8	Main endpoints of the metrics server.	42
4.9	Grafana dashboards of the administration tooling.	44
4.10	Views of the configuration frontend.	45
4.11	Configuration keys exposed by the configuration frontend.	46
5.1	Fulfillment of the functional requirements by the prototype.	51
6.1	Parts of the prototype.	57
6.2	Possible directions for future work.	58
6.3	Tools used during the project	68
A.1	Engagement Metrics	70
A.2	Retention Metrics	71
A.3	Communication Metrics	71
A.4	Acquisition & Growth Metrics	71
A.5	Feature Adoption Metrics	72
A.6	Technical / Performance Metrics	72
A.7	Social Graph Metrics	72
A.8	Notification & Reachability Metrics	73
A.9	Comparison of Analytics Platforms	73
A.10	Actual K-anonymity values at different number of submitted batches where the expected average k-anonymity is 10.	79

A.11 Actual K-anonymity values at different number of submitted batches where the expected minimum k-anonymity is 10.	79
--	----

List of Figures

2.1	Example of k-anonymity with $k = 3$. In the original data, every record has a unique combination of quasi-identifiers (age and ZIP code) and could be singled out. After <i>generalization</i> , the records collapse into two groups (highlighted) that each contain three identical records, so any individual now blends in with at least two others.	4
3.1	Maximum tolerable entropy budget $H_{\max}$ as a function of opt-in rate r for selected k-anonymity targets $k \in \{1, 2, 3, 4, 5, 10, 20, 50\}$, computed for a fixed user base of $N = 12'000'000$. The dashed vertical line marks the assumed opt-in rate of 20 %, and the filled circle marks the assumed operating point ($r = 20\%$, $k = 10$), yielding $H_{\max} \approx 17.8$ bits.	10
4.1	Alluvial diagram of the “Copy Poll” metric pipeline. The colored key denotes the four buckets a raw count is assigned to.	32
4.2	C4 Container Diagram displaying system components	35
4.3	UML overview of package architecture	37
4.4	UML view of where the package lives in the Threema desktop app and how it communicates	37
4.5	Database entity relation diagram	38
4.6	Sequence of a telemetry submission, from collection on the client to visualization.	48
5.1	Difference in network packet size when transmitting one and six metrics respectively (370 vs. 831 bytes)	50
A.1	Mockup of the Metrics Settings in Threema Desktop	75
A.2	Entity-Relationship Diagram for the Metrics System	76
A.3	K-anonymity values for different methods as a function of the number of submitted batches.	80
A.4	4-subset k-anonymity with an even distribution	82
A.5	4-subset k-anonymity with a uneven distribution	82
A.6	3-subset k-anonymity with a uneven distribution	83
A.7	Comparison of the available budget models for the entropy calculation . .	84
A.8	Popup Design Option 1	86
A.9	Popup Design Option 2	87

A.10 Popup Design Option 3	87
A.11 The implemented metrics setting in the privacy tab of the Threema Desktop application.	88
A.12 Landing page of the configuration frontend, linking to the two functional views.	89
A.13 Metric weights view, with one card per metric type and a slider for the recording weight.	90
A.14 Privacy budget view in the simple budget model.	90
A.15 Privacy budget view in the average-k budget model, which adds the subset size control.	91
A.16 Privacy budget view in the minimum-k budget model, which additionally exposes the minimum-k assumptions.	92
A.17 Metrics dashboard, showing the distribution of each metric across its buckets and the development of the submitted metrics over time.	93
A.18 Privacy parameters dashboard, visualizing the various k-anonymity calculations of the collected data.	94
A.19 Config keys timeline dashboard, showing the history of configuration changes.	95
A.20 Send requests and batches dashboard, monitoring the volume of received send requests and metric batches over time.	96
A.21 Metrics correlation analysis dashboard, visualizing the correlation between different metrics and how it develops over time.	97
A.22 The automatically generated OpenAPI documentation of the metrics server.	98

Tool Usage Collection

Table 6.3 gives an overview of the used tools during the project, categorized by area.

Table 6.3: Tools used during the project

Area	Tools
Research and information gathering	Google, Google Scholar, IEEE Library, Springer, NotebookLM
Idea finding and brainstorming	ChatGPT, Claude
Coding	Visual Studio Code, Neovim, ChatGPT, Claude
Editorial work	cspell, L ^A T _E X, Claude
Teamwork and project management	Outlook, Teams, GitLab, YouTrack, Discord, Threema
DevOps	Docker, GitLab

Acronyms

FADP Federal Act on Data Protection of 25 September 2020 (235.1)

GDPR General Data Protection Regulation of 27 April 2016 ((EU) 2016/679)

MoSCoW Must have, Should have, Could have, Won't have

Appendix A

Additional Artifacts

A.1 List of Metrics

A.1.1 Engagement

Table A.1: Engagement Metrics

Metric	Source
Daily Active Users (DAU)	[Amp25]
Monthly Active Users (MAU)	[Amp25]
DAU/MAU Ratio (Stickiness)	[Goo26d]
Session Frequency	[Amp26f]
Session Duration	[Amp26f]
Messages Sent per Session	[LH08]
Messages Sent per User per Day	[LH08]
Message Reciprocity Rate	[CAM ⁺ 23]
Response Time	[LWPS18]

A.1.2 Retention

Table A.2: Retention Metrics

Metric	Source
Day 1 / Day 7 / Day 30 Retention Rate	[Amp26e]
Churn Rate	[Amp24]
User Resurrection	[Amp26d]
User Lifecycle	[Amp26d]

A.1.3 Communication

Table A.3: Communication Metrics

Metric	Source
Message Volume	[SPHS22]
Message Type Distribution	[SSHTG15]
Conversation Depth	[LH08]
Group vs. Direct Message Ratio	[SSHTG15]
Broadcast / Notification Open Rate	[Air26]

A.1.4 Acquisition & Growth

Table A.4: Acquisition & Growth Metrics

Metric	Source
New User Registrations	[Goo26a]
Referral Rate	[Mix26a]
Funnel Conversion Rate (Install $\rightarrow$ Registration)	[Amp26c]
Source / Channel Attribution	[Goo26b]

A.1.5 Feature Adoption

Table A.5: Feature Adoption Metrics

Metric	Source
Feature Usage Rate	[Amp26b]
Search Usage Frequency	[Goo26c]
Notification Opt-in / Opt-out Rate	[Air26]
Funnel Conversion Rate (Profile Completion)	[Amp26c]

A.1.6 Technical / Performance

Table A.6: Technical / Performance Metrics

Metric	Source
Message Delivery Rate	[One25]
Crash Rate / Error Rate	[Goo25b]
App Load Time	[Goo25a]
Push Notification Delivery Rate	[One25]

A.1.7 Social Graph

Table A.7: Social Graph Metrics

Metric	Source
Contacts per User	[MMG ⁺ 07]
Active Contacts (last 30 days)	[MLCM13]
Group Membership Number	[SSHTG15]
Network Densification	[LKF07]

A.1.8 Notification & Reachability

Table A.8: Notification & Reachability Metrics

Metric	Source
Push Notification CTR	[One25]
Notification Backlog & Attentiveness	[PdOKO14]
Time-to-Open	[Cle26]

A.2 Comparison of Analytics Platforms

Table A.9: Comparison of Analytics Platforms. Compiled from the official documentation of Amplitude [Amp26a], Fullstory [Ful26], and Mixpanel [Mix26b].

Metrics Category	Data Point	Amplitude	Fullstory	Mixpanel
User Identity & Demographics				
Identity	User ID / Device ID	✓	✓	✓
	Anonymous ID	✓	✓	✓
	User Properties (Name, Email)	✓	✓	✓
	IP Address	✓	✓	✓
Location	City, Region, Country	✓	✓	✓
	DMA / Metro Code	✓	✗	✗
	Language / Locale	✓	✓	✓
Device & Technical Context				
Device	Device Type (Mobile/Desktop)	✓	✓	✓
	Device Model / Manufacturer	✓	✓	✓
	OS Name & Version	✓	✓	✓
	Browser Name & Version	✓	✓	✓
	Screen Height/Width/DPI	✓	✓	✓
	Carrier / Network Provider	✓	✓	✓
Connectivity		Continued on next page		

Table A.9 – continued from previous page

Metrics Category	Data Point	Amplitude	Fullstory	Mixpanel
	Connection Type (WiFi/4G)	✓	✓	✓
	App Version / Build Number	✓	✓	✓
Behavioral & Interaction Data				
Engagement	Session Duration	✓	✓	✓
	Session Frequency	✓	✓	✓
	Time Since Last Seen	✓	✓	✓
	Active Days (DAU/MAU)	✓	✓	✓
	Page Views / Screen Views	✓	✓	✓
Interaction	Clicks / Taps	✓	✓	✓
	Form Changes / Abandonment	✓	✓	✓
	Text Input Changes	✓	✓	✓
Acquisition & Attribution				
Traffic Source	UTM Parameters	✓	✓	✓
	Referrer URL / Domain	✓	✓	✓
	Initial vs. Current Source	✓	✓	✓
Performance & Experience Quality				
Frustration	Rage Clicks	✓	✓	✓
	Dead Clicks	✓	✓	✓
	Thrashed Cursor	✗	✓	✗
Technical	JavaScript Errors	✓	✓	✗
	Network Request Failures	✓	✓	✗
	Page Load Times	✓	✓	✗
	App Crashes	✗	✓	✗
Visual Experience & Replay				
Replay	Session Replay	✓	✓	✓
	Heatmaps (Click/Scroll)	✓	✓	✓
	Live Co-browsing	✗	✓	✗
	Mobile App Replay	✓	✓	✓
	AI Session Summary	✓	✓	✓

Continued on next page

Table A.9 – continued from previous page

Metrics Category	Data Point	Amplitude	Fullstory	Mixpanel
Advanced Analysis				
Analysis	Metric / KPI Trees	✓	✗	✓
	Root Cause Analysis	✓	✓	✓

A.3 Mockups

A.3.1 Threema Desktop

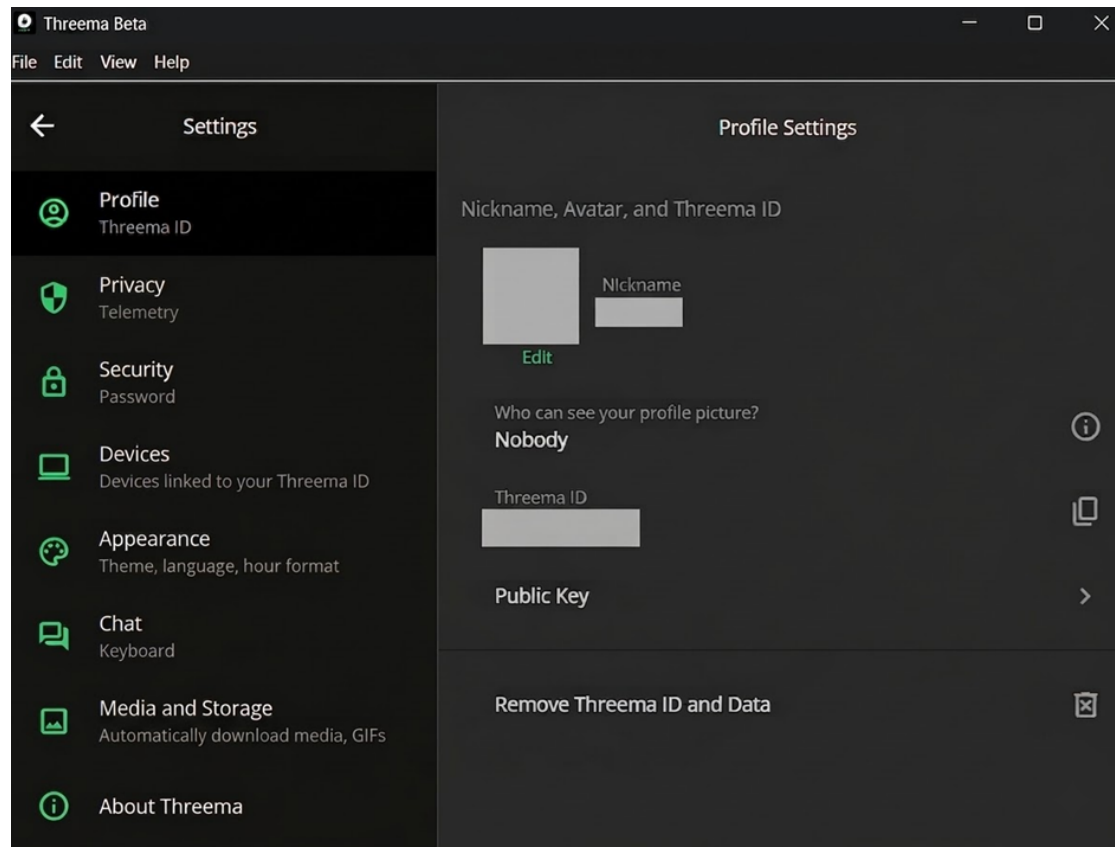


Figure A.1: Mockup of the Metrics Settings in Threema Desktop

A.4 Models

A.4.1 Entity Relationship Diagram

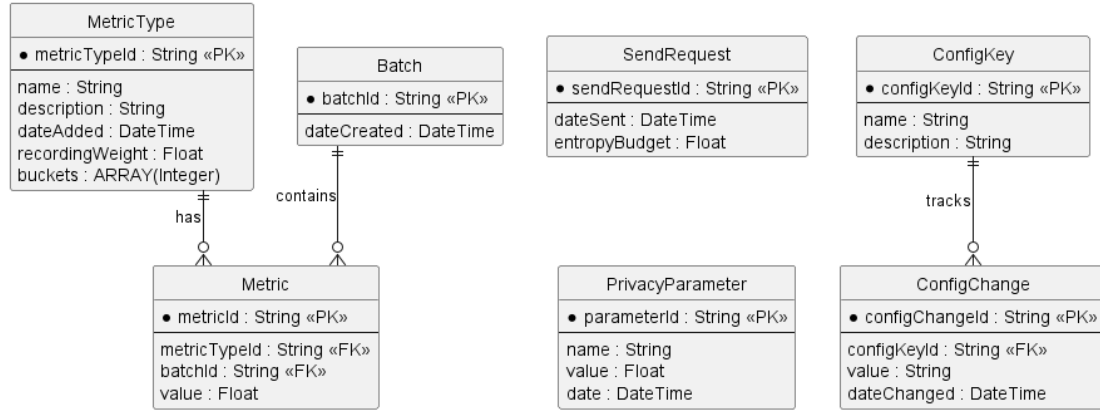


Figure A.2: Entity-Relationship Diagram for the Metrics System

A.4.2 Threat Model

To understand the potential risks that come with collecting telemetry data, we define a threat model. We use the LINDDUN model as it addresses the main concerns of our project: Anonymity and Unlinkability. We only focus on categories relevant to us.

A.4.2.1 Assets

The primary assets to protect are:

- **User Anonymity:** No individual user should be identifiable within the dataset.
- **Unlinkability:** Multiple telemetry submissions should not be linkable to the same user.
- **Data Integrity:** Submitted metrics should reflect genuine usage and not be manipulable.

A.4.2.2 Actors

- **Users:** Individuals using the Threema app who may opt-in to telemetry. They are trusted with telemetry information they generate, but not with data of any other users.
- **Threema GmbH:** The organization collecting and analyzing the telemetry data. While they are generally trusted to handle metrics data, our system aims to limit their potential to misuse the data by design.

- **Attackers:** External entities attempting to re-identify users or extract sensitive information from the dataset or otherwise. They are not trusted.

A.4.2.3 LIDDUN Threat Analysis

Linking *Threat:* Multiple telemetry submissions or metric combinations can be linked together to form a unique user profile.

Attack Vectors:

- Correlation of metric values across submissions. **Mitigation:** Entropy control to limit uniqueness of metric combinations.
- Network-level metadata (e.g., packet size, IP address). **Mitigation:** No storage of IP addresses and padded message lengths.

Identifying *Threat:* A user can be re-identified from a unique combination of metric values, or in the case of an insider attack, with correlating data from app usage which Threema GmbH has access to outside the telemetry system.

Attack Vectors:

- High entropy metric combinations (fingerprinting). **Mitigation:** Review of metrics once collected to ensure buckets are set appropriately.
- Using known data points to match a metric to a user (matching). **Mitigation:** Aiming for high k-anonymity value to ensure a metric combination corresponds to many users.
- Rare or extreme values (outliers). **Mitigation:** Bucketization to reduce granularity and control outliers.

Non-repudiation *Threat:* Being able to submit metrics as another user.

Attack Vectors:

- Submitting metrics with a forged user identifier. **Mitigation:** No user identifiers are used in the telemetry system, so this attack vector is not applicable.
- Submitting false metrics to skew the data. **Mitigation:** This is a potential risk which we have identified in subsection 2.3, but will not be mitigated in the scope of this thesis.

Detecting *Threat:* An attacker can detect whether a specific user is using the application or participating in telemetry.

Attack Vectors:

- Network-level metadata (e.g., IP address, packet size). **Mitigation:** No storage of IP addresses and padded message lengths.

Data Disclosure *Threat:* Sensitive information about user behavior is revealed through collected metrics.

Attack Vectors:

- Overly granular metrics. **Mitigation:** Bucketization to reduce granularity.
- Highly private metrics (e.g., message content). **Mitigation:** No collection of highly private metrics.

Unawareness *Threat:* Users are not aware of the data being collected or its implications. **Attack Vectors:**

- Lack of clear communication about telemetry. **Mitigation:** Pop-up with clear information is presented to users when asked to opt-in.
- Hidden option to opt-out. **Mitigation:** Clear and easily accessible opt-out option in the app settings.

Non-compliance *Threat:* The system violates data protection regulations such as GDPR or FADP.

Attack Vectors:

- Collection of data that can be linked to individuals. **Mitigation:** Strict adherence to anonymization principles.
- Lack of valid user consent. **Mitigation:** Clear opt-in process in combination with an Open-source client, making it possible to verify this without audits by external companies.

A.5 Simulated Testing

A.5.1 Basic K-anonymity Testing

To validate the calculations made in Section 3.2.2, we conduct some simulations without the more advanced components of the system (random sampling & dynamic budget). We calculate k-anonymity for a dataset of generated metric batches and check if the k-anonymity values are in line with our expectations based on the design of the system. We generate enough batches to reach a k-anonymity goal of 10, using the Equation 3.5. Then, we calculate the k-anonymity in a traditional sense (minimum number of batches which have the same combination of metric values) as well as the average k-anonymity (average group size across all groups). The results are shown in Table A.10.

Table A.10: Actual K-anonymity values at different number of submitted batches where the expected average k-anonymity is 10.

Metrics per batch	Batches	Min. k-anonymity	Avg. k-anonymity
2	160	4	11.1
3	640	2	11.1
4	2560	3	11.0
5	10240	1	11.0

For these tests, all buckets for all metrics were within 5% of the optimal distribution (random, even distribution), and the average k-anonymity values are in line with our expectations based on the design of the system, which is a good indication that the system behaves as expected in this basic case. However, the minimum k-anonymity, which follows the more traditional definition of k-anonymity, is still much lower than the average k-anonymity, which indicates that outliers in the dataset are not well protected. To alleviate this, we use the adjusted formula from Equation 3.6 to try and predict k-anonymities for the most vulnerable user. The results of this test can be seen in Table A.11.

Table A.11: Actual K-anonymity values at different number of submitted batches where the expected minimum k-anonymity is 10.

Metrics per batch	Batches	Min. k-anonymity	Avg. k-anonymity
2	301	8	19.9
3	1356	10	22.3
4	5965	9	24.4
5	27350	10	27.7

Using the adjusted formula, we can predict the *minimum k-anonymity value* to a level of accuracy which we deem acceptable for our use case, as there will always be some random variation in the actual minimum k-anonymity due to random distribution of users across buckets. This shows that our basic system design works, which allows us to move on with testing the more advanced components of the system.

A.5.2 K-anonymity Methods with Random Sampling

To evaluate the different k-anonymity methods with nullable values from the random sampling approach we have presented in Section 3.3.2, we have constructed the following test:

1. Generate *SendRequests* and *Batches* for 5000 users sequentially, with the same metrics selection algorithm as implemented in the mobile package.
2. Bucket values are generated with a **even distribution**.
3. Re-calculate k-anonymity values for all presented methods at an interval.
4. Evaluate how k-anonymity behaves for each method as the number of users increases.

We conduct this test and plot the results in Figure A.3.

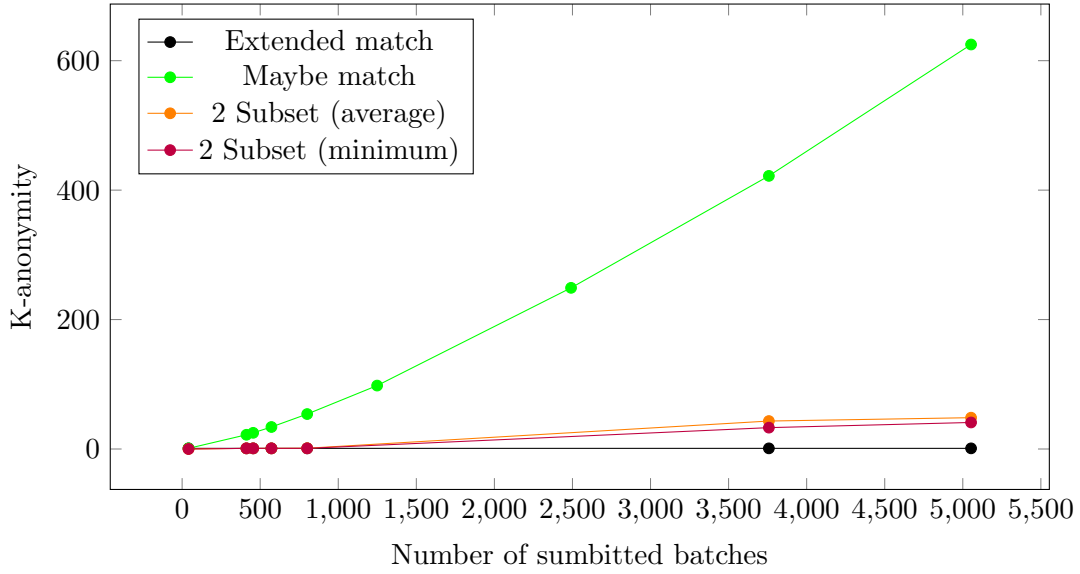


Figure A.3: K-anonymity values for different methods as a function of the number of submitted batches.

- The basic match (not shown on the graph) method returns a k-anonymity of 0 for all batches, as they all contain at least one nullable value and are therefore deleted. This is not a useful result, so we also do not consider it for further evaluation.
- The extended match method returns a k-anonymity of 1 for almost all batches, as a exact combination of reported and non-reported metrics as well as values for all reported metrics is unique for almost all users. As discussed in Section 3.3.2, we do not consider the non-reported metrics to be useful for re-identification, so we do not consider this method for further evaluation.
- Due to the high number of nullable values, the maybe match results in high k-values, because many rows match with each other due to treating the NULL values as wildcards. Additionally, the computation of this method is significantly more expensive than the other methods as the groups created are not transitive. Also,

Ciglic et al. [CEK14] conclude that this method is not safe from linking attacks. For these reasons, we do not consider this method for further evaluation.

The remaining **subset method** is the only method which provides meaningful k-anonymity values. It provides lower k-anonymity values for the same amount of batches compared to the Basic K-anonymity test, which is expected because not all metrics are reported in all batches, leading to more groups with lower k-anonymity values.

A.5.3 Subset Method with Large Datasets

To test the performance of the subset method in scenarios which are closer to the real world, we conduct the following test:

1. Generate *SendRequests* and *Batches* sequentially. Each batch contains 4 metrics.
2. Bucket values are generated with the following distributions:
 - **Test 1:** All buckets for all metrics are within 2% of the optimal distribution (random, even distribution).
 - **Test 2:** Bucket metrics are uneven to simulate misplaced buckets. We follow a normal distribution with a mean of 1 and standard deviation of 2, floored to the nearest bucket value.
3. Re-calculate k-anonymity values for various methods at an interval.
4. Evaluate how k-anonymity behaves for each method as the number of users increases, and how it compares to the expected k-anonymity values based on the design of the system.

We conduct **Test 1**, which represents a balanced bucket distribution for all metrics, as described above. The results are shown in Figure A.4. The measured average and minimum k-anonymity values are roughly in line with the predicted values, with the average k-anonymity always being about 1 higher than the predicted value, and the minimum k-anonymity data points always being within 5 of the predicted value.

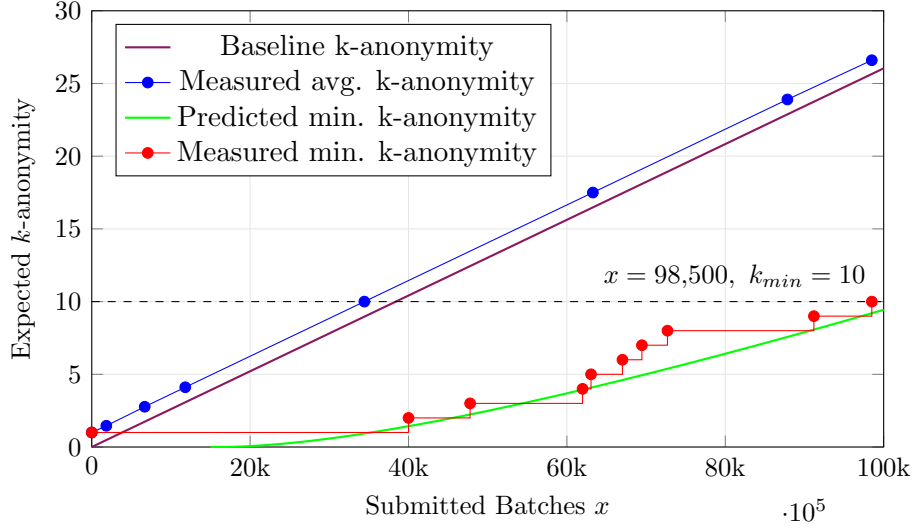


Figure A.4: 4-subset k -anonymity with an even distribution

To simulate a more realistic scenario, we conduct **Test 2**, which represents a more uneven bucket distribution for all metrics, as described above. With this uneven value generator, we have a distribution in buckets of about 40%, 23%, 18% and 19% for buckets 1-4 respectively. The results are shown in Figure A.5 and Figure A.6.

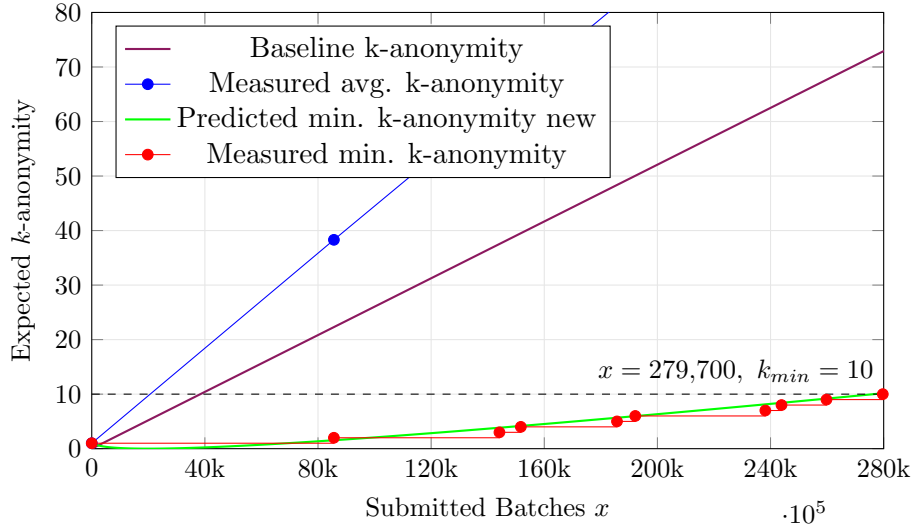


Figure A.5: 4-subset k -anonymity with a uneven distribution

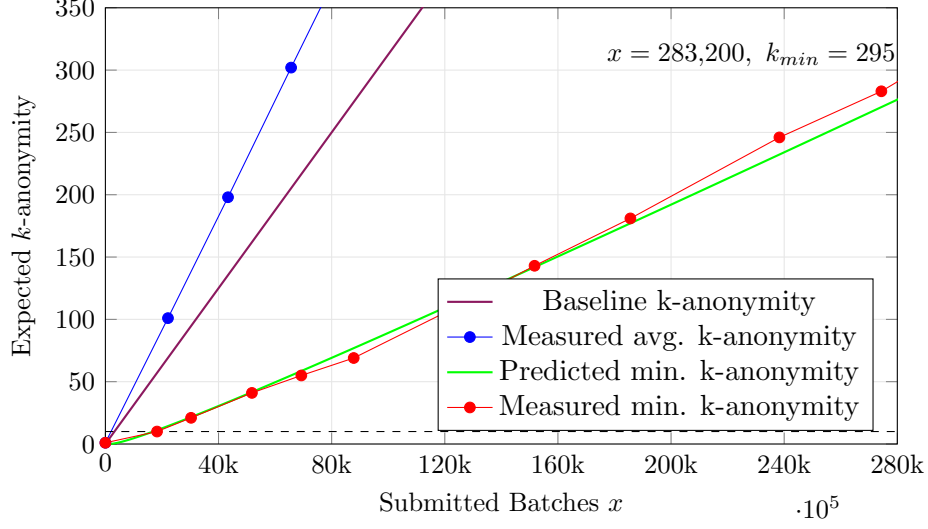


Figure A.6: 3-subset k -anonymity with a uneven distribution

From these tests, we can see that our formulas for predicting average and minimum k -anonymity (defined in Section 3.3.3) are performing as expected. We can see that the *Baseline k -anonymity* is lower than the actual datapoints, which is expected in a scenario with unbalanced buckets like this, as described in the explanation of the formula. In a real-world scenario, this should indicate that the buckets are not distributed evenly and re-distribution of buckets should be considered. The *Predicted min. k -anonymity* is roughly in line with the actual data points, which indicates that it can be used as a good predictor for minimum k -anonymity values in practice.

A.5.4 Alternate Entropy Calculation Methods

To test and visualize how different budget models calculate the entropy, we conduct the following test:

1. Set the desired budget model as well as the corresponding parameters: Target $K = 10$, Subset Size 3, Rarest Subset Probability: 0.005832, Rare Combination Count: 1, Total Metrics: 6, Average Entropy per Metric: 2.
2. Generate and send *SendRequests*
3. Monitor how the entropy budget changes with each sent batch

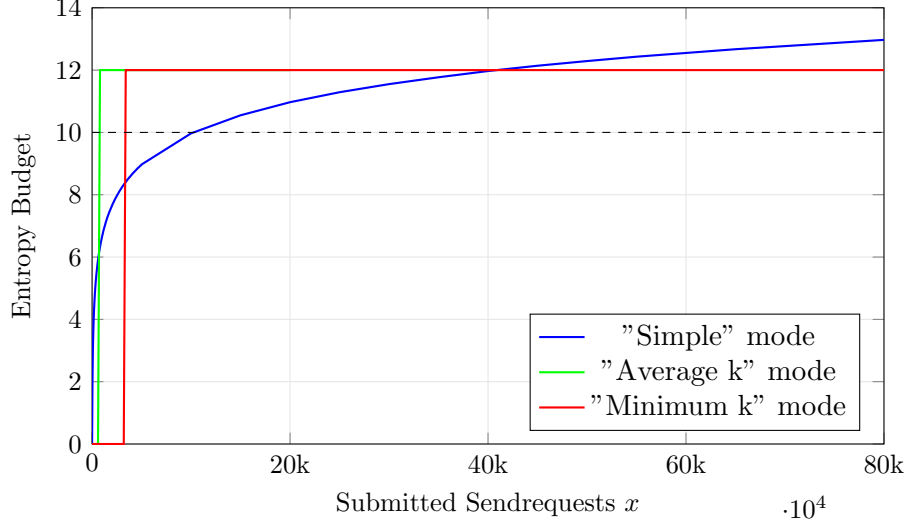


Figure A.7: Comparison of the available budget models for the entropy calculation

From this test, we can see that the simple mode model provides a gradually increasing budget as expected. The average and minimum k model on the other hand jump to $H = 12$ quickly (*Average K Mode at 640 active Users, Minimum K mode at 3400 active Users*). This can be explained in the following way: These two functions are optimizing the provided formulas for k_{base} and k_{min} (Equation 3.5 and 3.6) respectively as described in Appendix A.6.3. The subset method for calculating the k-anonymity these modes are aiming for is actually producing higher k-anonymity values the more metrics are sent in a batch. This is because the more metrics are sent in a batch, the more combinations of metric values there are, which leads to more batches being in the same combination of metric values. This leads to higher k-anonymity values using the subset method.

A.6 Formulas and Calculations

A.6.1 Calculating System State

In order to calculate entropy budgets, we need to know the state of the system.

1. **Users in the System:** The total number of SendRequest submissions received within a sliding time window.
2. **Average Entropy, Total Metrics:** The system reads the average entropy of all metrics as well as total number of metrics from the provided *metrics-definition.json* file.

A.6.2 From k_{base} to k_{min}

Starting from the definition of k_{base} (Equation 3.5), we can isolate the expected count in a single bucket:

$$\mu = k_{base} \cdot p_{min} \cdot b^{M_S} \quad (\text{A.1})$$

which simplifies, by substituting Equation 3.5, to:

$$\mu = \frac{n_B \cdot p_{min} \cdot \binom{M_R}{M_S}}{\binom{M_T}{M_S}} \quad (\text{A.2})$$

The total number of buckets sharing this minimum probability p_{min} (assuming comparable distributions) across all metric subsets is:

$$N = c_{min} \cdot \binom{M_T}{M_S} \quad (\text{A.3})$$

Following these steps, we add corrective terms (extreme value theory) to the Poisson distribution to estimate the min

$$k_{min} \approx \mu - \sqrt{2\mu \ln(N)} + \frac{\ln(N)}{2} \quad (\text{A.4})$$

where the $-\sqrt{2\mu \ln(N)}$ term captures the downward shift (the more buckets you have, the more likely one is extremely empty), and $+\frac{\ln(N)}{2}$ corrects for the fact that buckets can only contain a fixed integer value, while the poisson distribution does not account for this.

A.6.3 Calculating privacy budget

To calculate the privacy budget for a given set of parameters, we use the following formulas:

A.6.3.1 "Simple" Mode

The `SIMPLE_MODE` privacy budget is calculated using the same formula as first proposed in Section 3.2.2.2, Equation 3.3.

$$H_{max} = \log_2 \left(\frac{\text{Number of opt-in users}}{\text{Target k-anonymity}} \right) \quad (\text{A.5})$$

A.6.3.2 Average and Minimum K-Anonymity Modes

Both `MODE_AVERAGE_K` and `MODE_MINIMUM_K` use a optimization loop to find the best budget.

Because a client can only send a whole number of metrics, using the formulas (Equation 3.5 and 3.6) directly creates fractional numbers that cannot be used in reality. To

solve this, the system tests whole numbers one by one. It starts at the maximum total metrics available (M_T) and counts down to the minimum allowed subset size (M_S). For each tested number of metrics (M_R), the system checks if the math is safe:

- **In Average k Mode:** It plugs M_R into Equation 3.5 to calculate the expected average group size (k_{base}).
- **In Minimum k Mode:** It calculates the expected size of the absolute rarest bucket (μ) and uses Equation 3.6 to find the worst-case group size (k_{min}).

The countdown stops as soon as it finds the highest number of metrics (M_R) that successfully meets the target ($k \geq k_{\text{target}}$).

A.7 Opt-in

A.7.1 Popup Designs

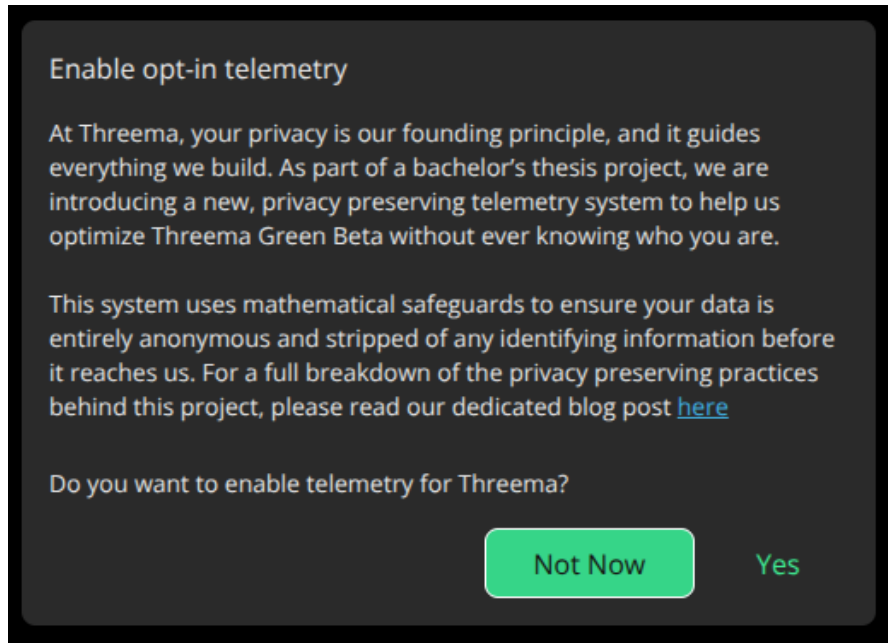


Figure A.8: Popup Design Option 1

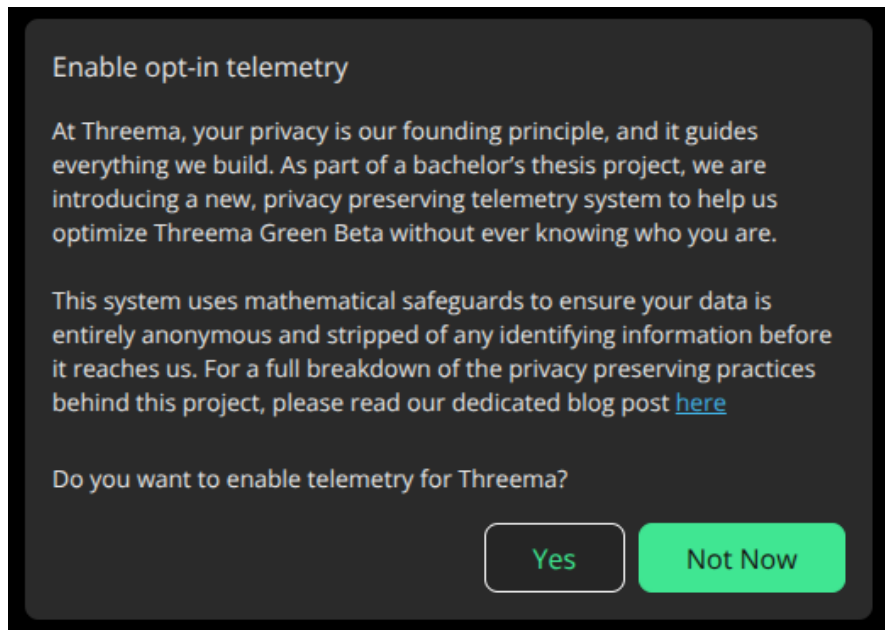


Figure A.9: Popup Design Option 2

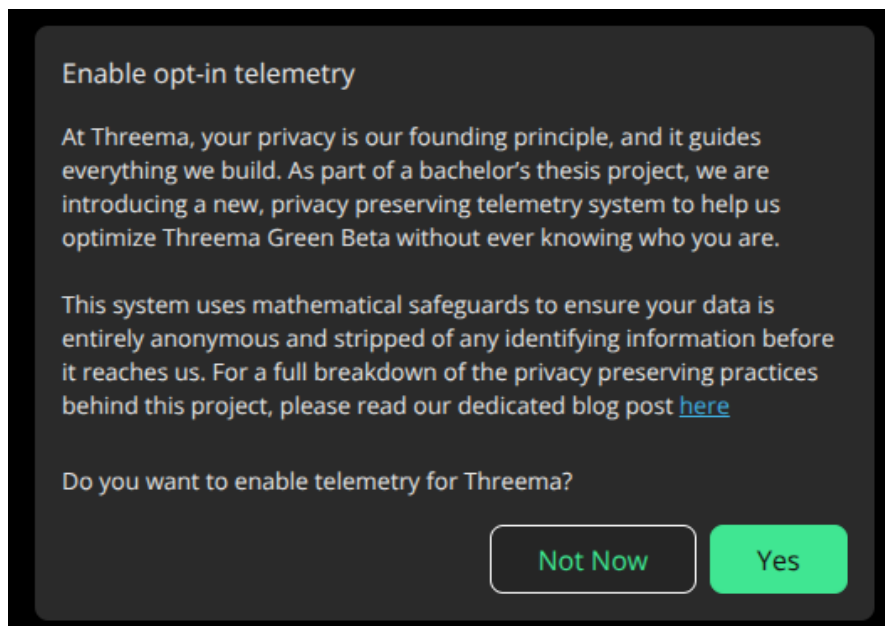


Figure A.10: Popup Design Option 3

A.7.2 Implemented Setting

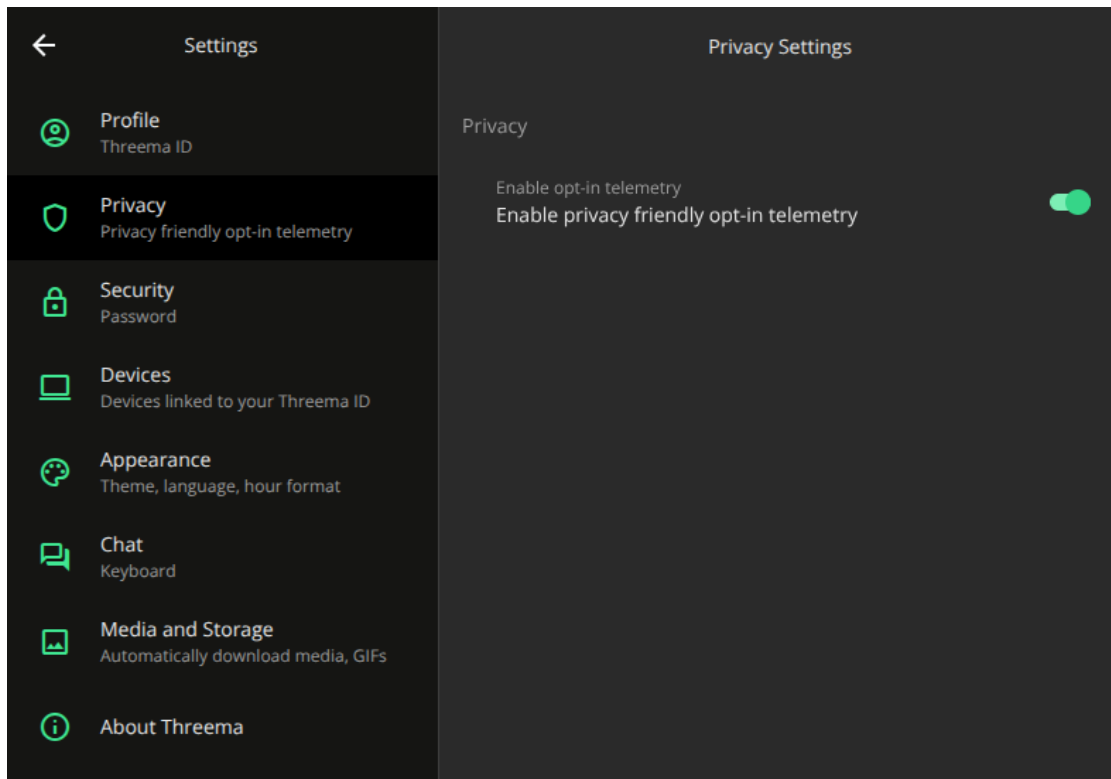


Figure A.11: The implemented metrics setting in the privacy tab of the Threema Desktop application.

A.8 Configuration Frontend Screens

This appendix collects screenshots of the configuration frontend described in [Section 4.11](#).

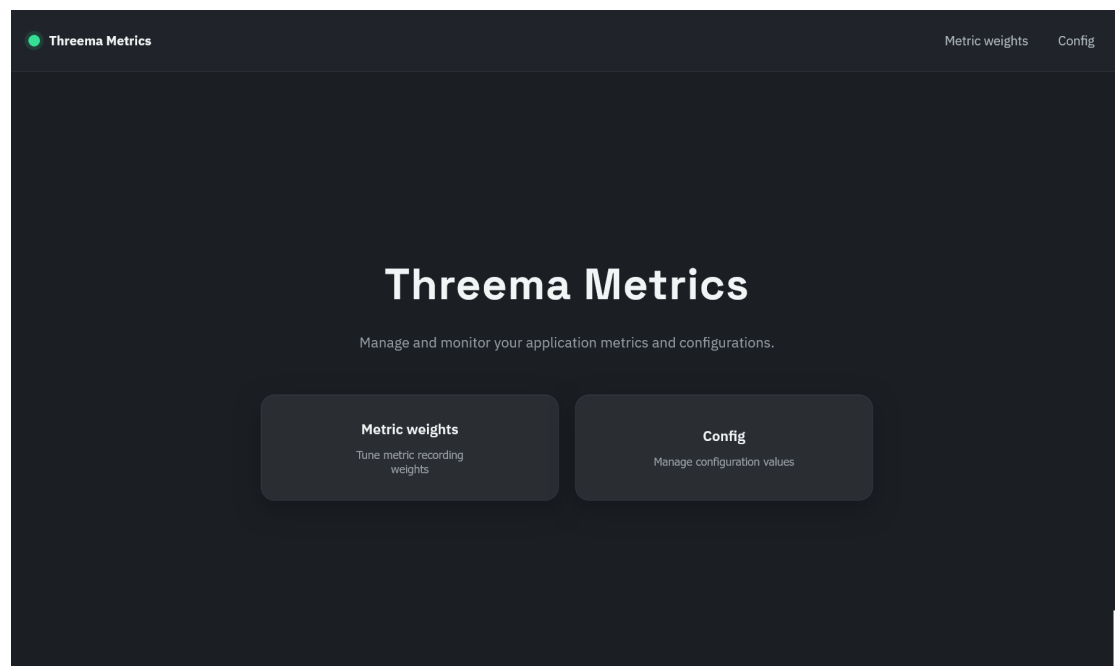


Figure A.12: Landing page of the configuration frontend, linking to the two functional views.

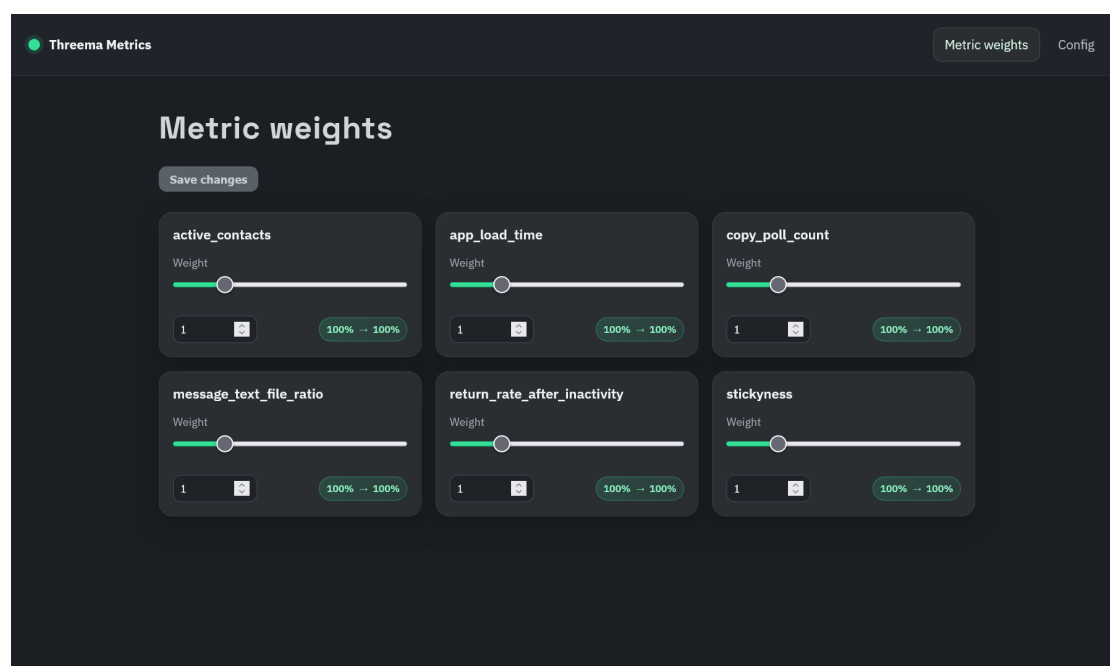


Figure A.13: Metric weights view, with one card per metric type and a slider for the recording weight.

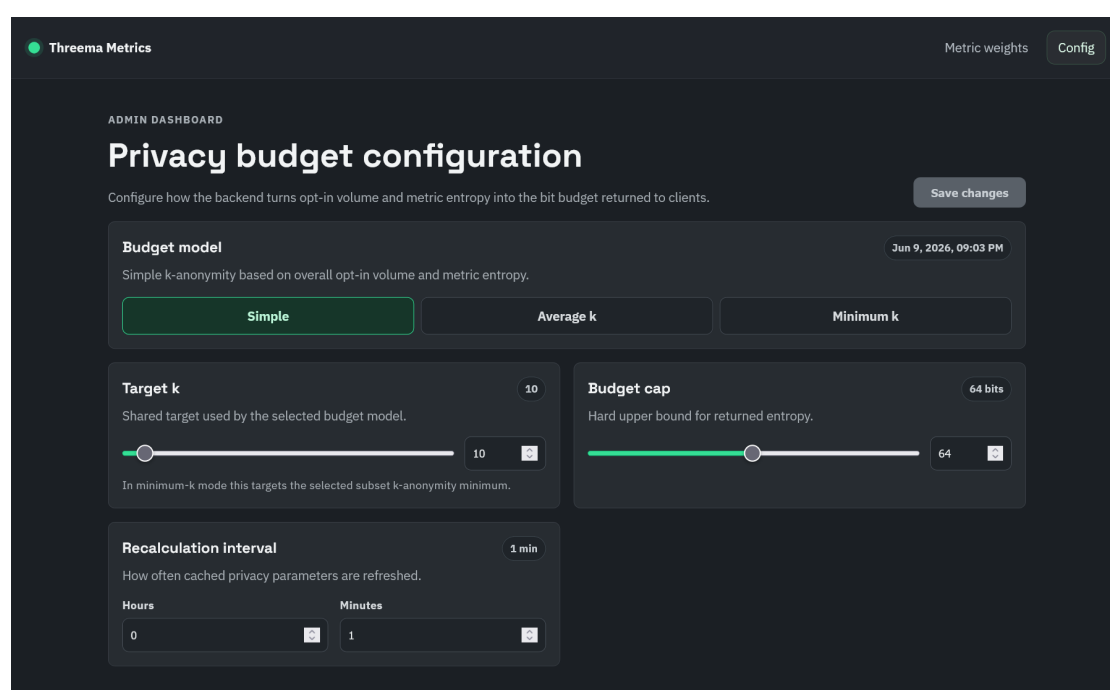


Figure A.14: Privacy budget view in the simple budget model.

Threema Metrics

Metric weightsConfig

ADMIN DASHBOARD

Privacy budget configuration

Configure how the backend turns opt-in volume and metric entropy into the bit budget returned to clients.

Save changes

Budget model

Jun 9, 2026, 09:29 PM

Predicts average subset k-anonymity from the metric bit budget.

SimpleAverage kMinimum k

Target k

10

Shared target used by the selected budget model.

10

In minimum-k mode this targets the selected subset k-anonymity minimum.

Budget cap

64 bits

Hard upper bound for returned entropy.

64

Subset size

3 metrics

Number of metric dimensions used for subset k-anonymity.

-

3

+

Recalculation interval

1 min

How often cached privacy parameters are refreshed.

Hours

Minutes

0

1

Figure A.15: Privacy budget view in the average-k budget model, which adds the subset size control.

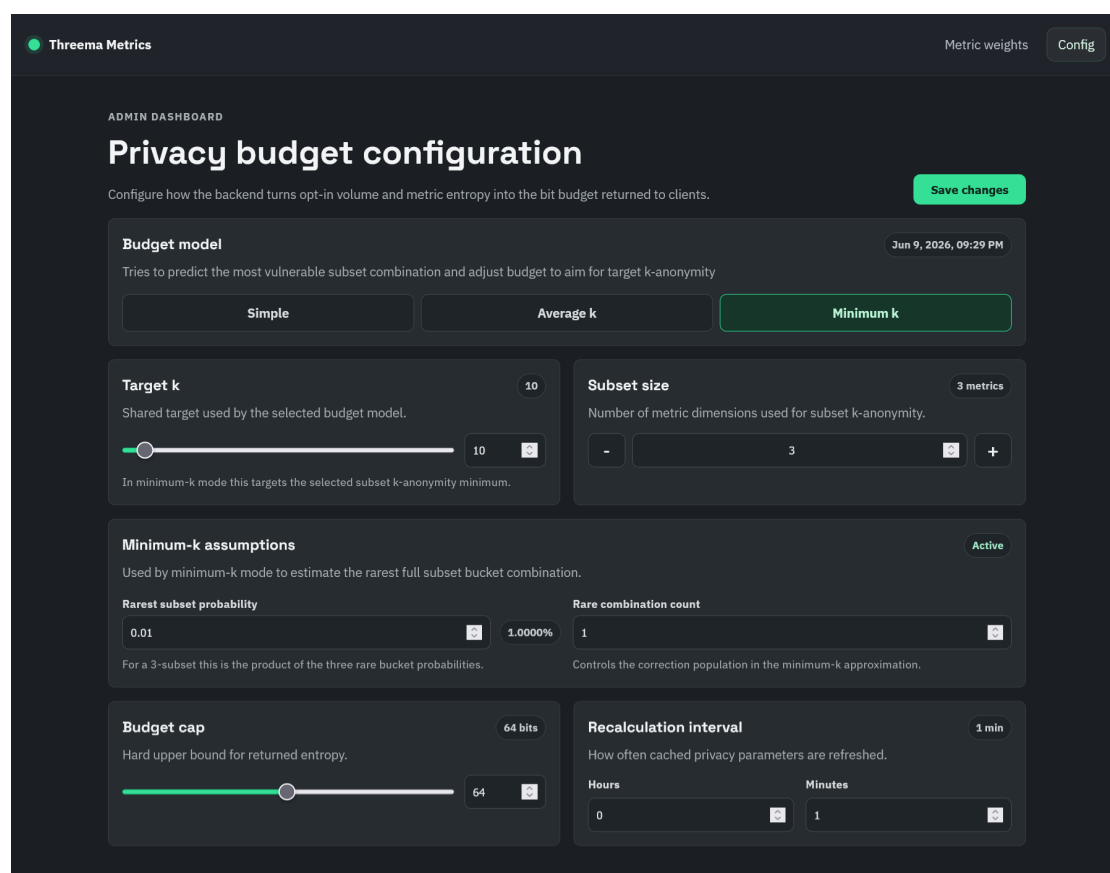


Figure A.16: Privacy budget view in the minimum-k budget model, which additionally exposes the minimum-k assumptions.

A.9 Grafana Dashboards

This appendix collects screenshots of the Grafana dashboards described in Section 4.10.

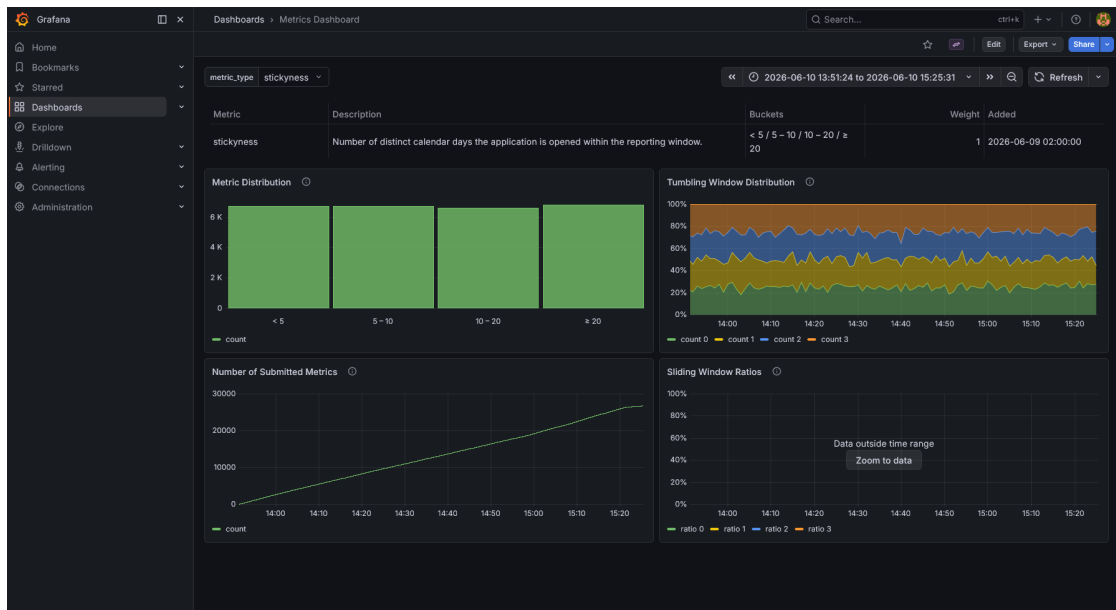


Figure A.17: Metrics dashboard, showing the distribution of each metric across its buckets and the development of the submitted metrics over time.

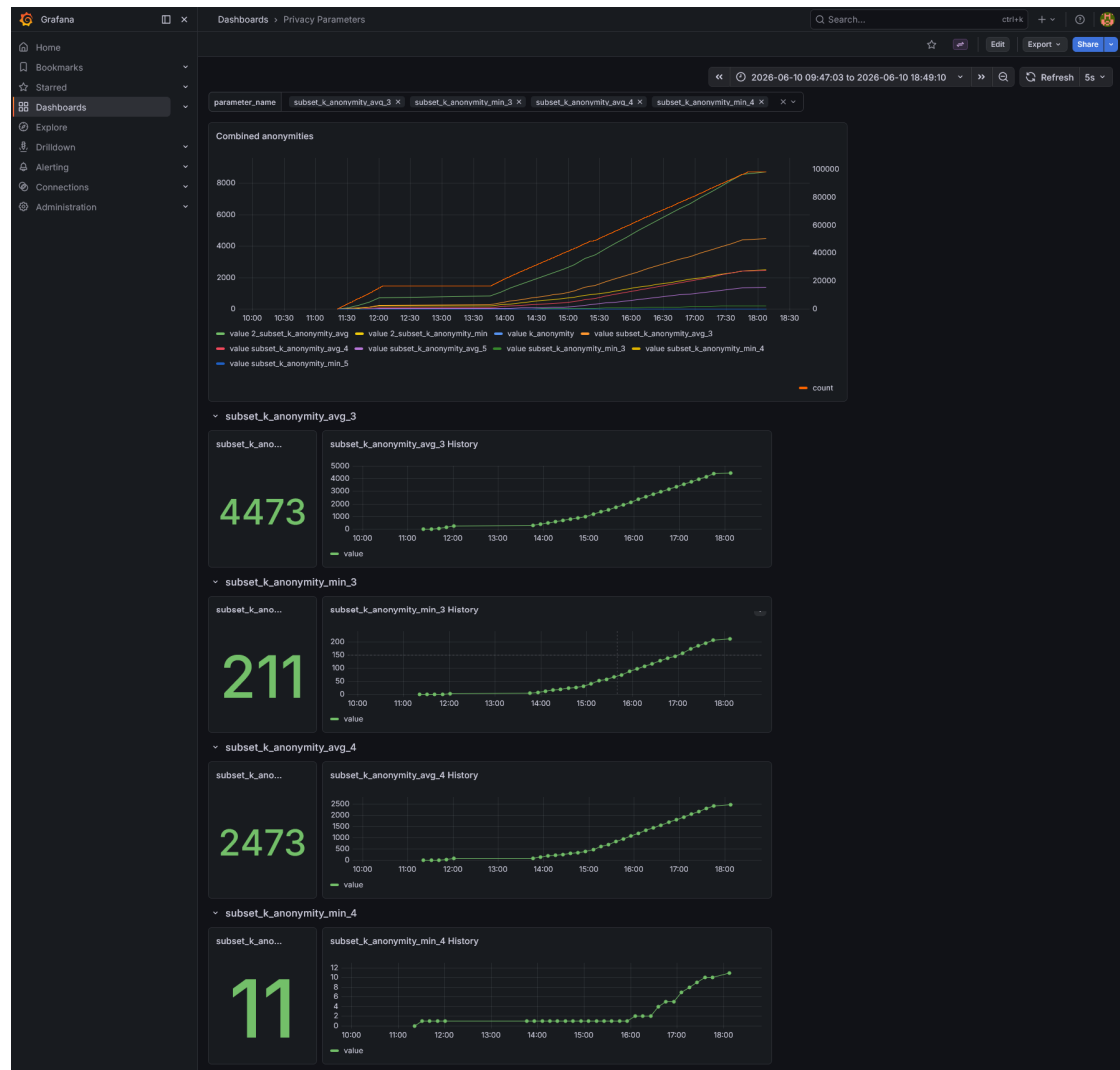


Figure A.18: Privacy parameters dashboard, visualizing the various k-anonymity calculations of the collected data.

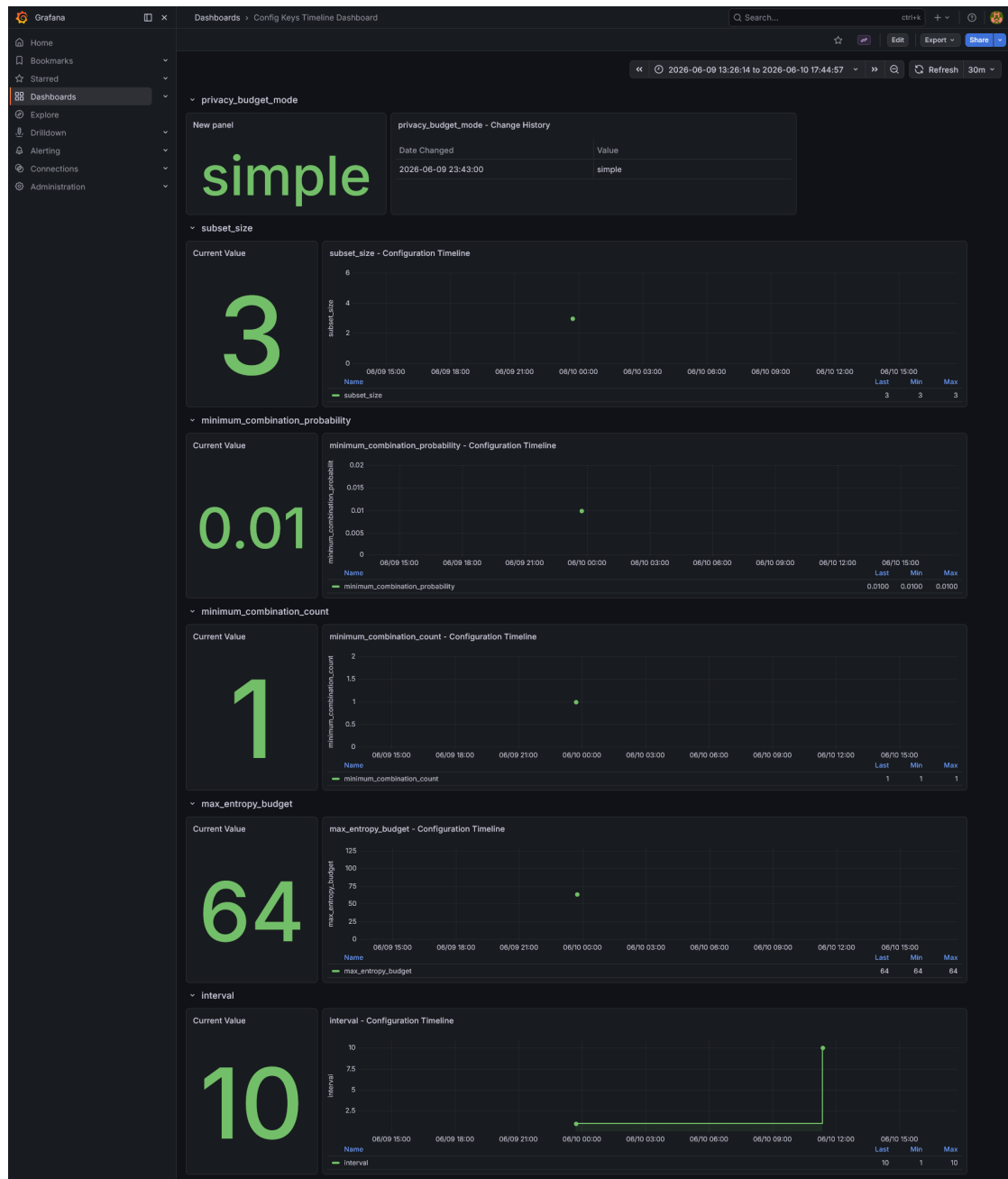


Figure A.19: Config keys timeline dashboard, showing the history of configuration changes.

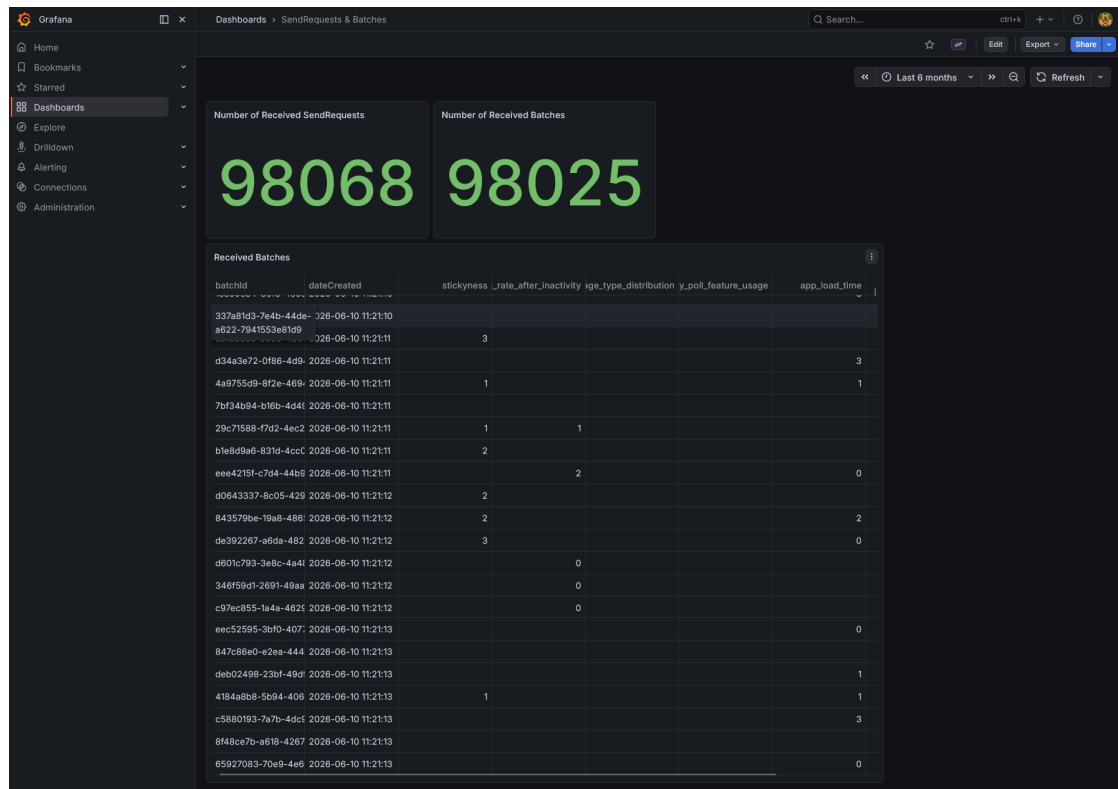


Figure A.20: Send requests and batches dashboard, monitoring the volume of received send requests and metric batches over time.

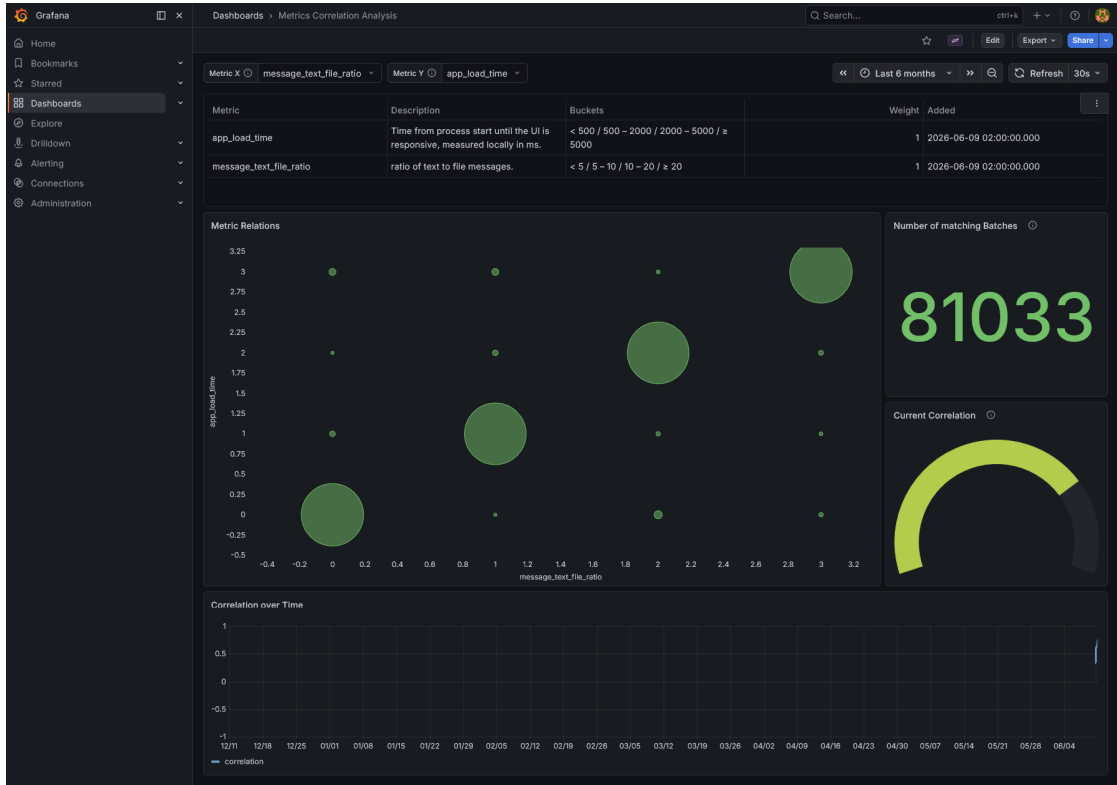


Figure A.21: Metrics correlation analysis dashboard, visualizing the correlation between different metrics and how it develops over time.

A.10 OpenAPI Specification

The metrics server (Section 4.8) automatically generates an interactive OpenAPI specification, which is served through its `/docs` endpoint. It documents every endpoint together with its request and response schemas and allows the API to be explored and tested directly in the browser, without any manually maintained documentation. The generated documentation is shown in Figure A.22.

Config Keys		^
POST	/config-keys/ Create Config Key	▼
GET	/config-keys/ Read Config Keys	▼
GET	/config-keys/with-values Read Config Keys With Values	▼
PUT	/config-keys/{config_key_id}/value Set Config Key Value	▼
Metrics		^
POST	/metrics/send-request/ Create Send Request	▼
POST	/metrics/batch/ Submit Metrics Batch	▼
Metric Types		^
GET	/metrictypes Get Metric Types	▼
PUT	/metrictypes/{metric_type_id} Update Metric Type Probability	▼

Figure A.22: The automatically generated OpenAPI documentation of the metrics server.

A.11 Example Metrics Definition File

```
[
  {
    "name": "stickiness",
    "description": "Number of distinct calendar days...",
    "id": "78a07cfb-3e3f-48db-ae6f-27bb3e27a03b",
    "buckets": [5, 10, 20],
    "enabled": true,
    "differentialPrivacyEnabled": true
  },
  {
    "name": "return_rate_after_inactivity",
    "description": "Time between the end of the last...",
    "id": "98fccdeb-5c39-4ea1-9894-d0d7815a11ef",
    "buckets": [180, 720, 2880],
    "enabled": true,
    "differentialPrivacyEnabled": true
  },
  {
    "name": "message_text_file_ratio",
    "description": "ratio of text to file messages.",
    "id": "d699d41e-a045-4118-b97c-62e1662f9f23",
    "buckets": [5, 10, 20],
    "enabled": true,
  }
]
```

```

    "differentialPrivacyEnabled": true
  },
  {
    "name": "copy_poll_count",
    "description": "Count_of_'Copy_Poll'_actions...",
    "id": "f5758fb0-3efc-4415-8981-b715a400e8ea",
    "buckets": [10, 20, 50],
    "enabled": true,
    "differentialPrivacyEnabled": true
  },
  {
    "name": "app_load_time",
    "description": "Time_from_process_start_until...",
    "id": "1f24c4e2-fa67-49b3-8c95-be038c165486",
    "buckets": [500, 2000, 5000],
    "enabled": true,
    "differentialPrivacyEnabled": true
  },
  {
    "name": "active_contacts",
    "description": "The_number_of_unique_contacts_the...",
    "id": "d1d56a7c-ecce-49c8-bb6e-24cf215f2b2d",
    "buckets": [5, 10, 20],
    "enabled": true,
    "differentialPrivacyEnabled": true
  }
]

```