USENIX Security '25 Artifact Appendix: How to Compare Bandwidth Constrained Two-Party Secure Messaging Protocols: A Quest for A More Efficient and Secure Post-Quantum Protocol

Benedikt Auerbach
PQShield

Yevgeniy Dodis
New York University

Daniel Jost
New York University

Shuichi Katsumata
PQShield and AIST

Rolfe Schmidt
Signal Messenger

30 May 2025 v0.0

A Artifact Appendix

A.1 Abstract

This artifact contains the code used to implement and evaluate the post-quantum ratcheting protocols described in the paper. The paper presents protocols using three KEM variants - Ratcheting KEM (RKEM), Unidirectional KEM (UniKEM), and Bidirectional KEM (BiKEM) - and using two sampling strategies - Opportunistic and Non-opportunistic - for a total of six protocols. The primary goal of these post-quantum secure messaging protocols is to provide post-quantum post compromise security (PCS), and we quantify this by measuring the size of vulnerable message sets ($|VulM|$) that can be recovered by an adversary after compromising one of the parties.

While this metric is natural, it also depends fundamentally on the underlying messaging behavior or the protocol participants. Among other issues, when the underlying messaging behavior is random, the size of $|VulM|$ uponcompromise will be a random variable. Without specifying a messaging behavior, these protocols are notcomparable: we can find a messaging behavior where any one of the six protocols produces the smallest $|VulM|$. Thus we simulate five messaging behaviors that approximate realistic usage by users of Signal Messenger, modeling balanced and unbalanced communication, as well as the behaviors of primary devices (cell phones that are usually online) and linked devices (often desktops that are online only when the user is at their desk). All of these behaviors are implemented in this artifact.

Finally, this artifact provides a top-level program that will perform a simulation of any of the protocols under any of the messaging behaviors and output statistics about the distribution of sizes of the resulting $|VulM|$. Scripts are also provided torun all combinations of protocols and messaging behaviors, then use the output data to reproduce the charts presented in the paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

The artifact is available on Zenodo at the following URL:

```
https://zenodo.org/records/15571277
```

A.2.3 Hardware dependencies

None.

A.2.4 Software dependencies

This artifact requires the following software dependencies:

- `Rust 1.88.0`: On Unix-like systems, the rust toolchain can be installed with `rustup` using the command:

  ```
  curl --proto '=https' --tlsv1.2 \\
    -sSf https://sh.rustup.rs | sh
  ```

  To install the specific version, you can run:

  ```
  rustup install 1.88.0
  rustup default 1.88.0
  ```

  but be aware that this will override the default version of Rust on your system, affecting other Rust projects.

- `Gnuplot`: On Debian Linux, run the command:

  ```
  sudo apt install gnuplot
  ```

On MacOS run the command:

```
brew install gnuplot
```

A.2.5 Benchmarks

None.

A.3 Set-up

With the Rust toolchain and Gnuplot installed, the artifact can be built by running the following command in the root directory of the project:

```
cargo build --release
```

This will compile the Rust code and produce an executable in the `target/release` directory.

A.3.1 Installation

Once compiled, no further installation is needed. The executable can be run to give basic usage information with the following command:

```
./target/release/usenix2025sims --help
```

A.3.2 Basic Test

The program can be used to run simulations of different protocols under different messaging behaviors described in the paper by executing the following command:

```
./target/release/usenix2025sims  \\
   [OPTIONS] [PROTOCOL] [BEHAVIOR] \\
   [KEMTYPE] [OUTPUT] [CHUNKSIZE]
```

where:

- `[OPTIONS]`: Optional command-line options, such as `-numsteps` to specify the number of time steps for the simulation.
- `[PROTOCOL]`: The protocol to simulate, one of `opp-rkem, rkem, opp-uni-kem, opp-bi-kem, uni-kem, bi-kem`.
- `[BEHAVIOR]`: The messaging behavior, one of `balanced, pingpong, random, desktop-desktop-disjoint, primary-to-desktop, desktop-desktop-overlap`.
- `[KEMTYPE]`: The key encapsulation mechanism type, either `fs-rkem` to us a Forward Secret Ratcheting KEM or `non-fs-rkem` for the non Forward secret RKEM variant.
- `[OUTPUT]`: The data to output. Use `hist` to output a histogram of vulnerable message set sizes.
- `[CHUNKSIZE]`: The size of the chunks to process, e.g., `small` for 32B chunks, `medium` for 128B chunks, pr `large` for 512B chunks.

For example, to simulate the Opportunistic RKEM protocol with 32B chunks under the primary-primary messaging behavior for 100,000 time steps, run the following command:

```
./target/release/usenix2025sims \\
  --numsteps 100000 opp-rkem balanced \\
  fs-rkem hist small
```

You will then find two CSV files:

- `default_bal_opprkem_hist.csv` contains the frequency and cumulative distributions of the sizes of the vulnerable message sets upon compromize of parties A and B throughout the simulation.
- `default_bal_opprkem_stats.csv` contains the mean, standard deviation, and deciles of the vulnerable message sets exposed by parties A and B throughout the simulation.

A.4 Evaluation workflow

Scripts are provided to reproduce all experiments and charts in the paper.

A.4.1 Major Claims

The following claims are made in the paper, and are supported by the experiments in this artifact:

(C1): As SM protocols supporting larger bandwidth allow larger chunks to be sent, $|VulM|$ becomes smaller. However, since the protocol cannot become secure while one party is offline, the benefit of allowing larger chunks diminishes as the overlap of the parties' online time becomes small.

(C2): The security of an SM protocol hinges on the most insecure party; if one party is compromised, then all messages sent during that time become vulnerable, regardless of the peer's corruption state. As such, unbalanced communications amplify the time it takes to recover from a compromise, making the average and variance of $|VulM|$ larger.

(C3): A compromised party that is offline cannot recover, and thus, the average and variance of $|VulM|$ increase. The primary-primary and overlapping desktop-desktop model produce similar cumulative probability of $|VulM|$ as both send a similar number of messages and are online at the same time enough to avoid blocking; For primary-desktop we see a higher average of $|VulM|$ because the communication is unbalanced and the online party will continue sending vulnerable messages while the other party is offline.

(C4): Opportunistic sending is consistently better (i.e., small average and variance of $|VulM|$) for small chunks, suggesting that wasting bandwidth is more harmful than sampling key material in advance. However, when the chunks become larger the benefit starts to diminish as they behave similarly. This is because as long as the party is offline, no recovery can happen — even if the online party finishes sending all information required to proceed to the next epoch secret.

(C5): Among the opportunistic protocols, Opp-RKEM and Opp-UniKEM perform similarly, and outperform Opp-BiKEM for all messaging behaviors except the "disjoint desktop-desktop" model.

A.4.2 Experiments

We provide a script that simulates all analyzed protocols undr all analyzed messaging behaviors, and produces the charts in the paper. These charts provide views of the generated data and support the claims above.

(E1): *[Main charts] [1 human-minute + 1 compute-minute + 4MB disk]: execute all protocols for all messaging behaviors and produce the charts in the paper.*

Execution: From the main directory, run the following commands:

```
cd charts/
./run\_tests.sh
```

Results: The figures from the main body of the paper will be generated in the directory `charts/figs/`. All charts generated by this script should match those in the paper exactly.

- To assess claim C1, the figure `fig4_primaryprimary_unikem_varchunksize.png` clearly shows that for the "primary-primary" messaging behavior which simulates two online cellphones, increasing the bandwidth limit and using larger message chunks leads to a clear reduction in the size of $|VulM|$. The figure `fig4_disjoint_dd_uniikem_varchunksize.png`, on the other hand shows the vulnerable message set size distributions of the same protocols, but under the"disjoint desktop-desktop" messaging behavior which simulates two Signal linked desktop devices that are never online at the same time. In this case, the advantages of sending larger messages disappear. These results are for the UniKEM protocol, but similar results can be seen for the other protocols.
- To assess claim C2, consider figures `fig5_unbal_uni_32.png` and `fig5b_unbal_rkem_32.png` which show the distributions of $|VulM|$ under balanced and unbalanced communication for the Opp-UniKEM and Opp-RKEM protocols respectively. For unbalanced communication we simulate situations where party A sends 9 times more messages than party B and vice versa. The figures show that the average and variance of $|VulM|$ are much larger for unbalanced communication. While not part of claim C2, the figures also bring out the inherent asymmetry of the Opp-UniKEM protocol, where security is more sensitive to a slowdown by party A than party B. This is because this protocol requires party A to transmit more data.
- To assess claim C3, consider figure `fig6a_online_overlap_katana_32.png`. Messaging behaviors that have both parties often online at the same time - the "primary-primary" and ""overlap desktop-desktop" models - yield similar cumulative probability distributions. The "primary-desktop" model, on the other hand, shows a higher average of $|VulM|$ because the online primary device is often sending messages to an offline desktop. The "disjoint desktop-desktop" model is even more extreme, as it simulates two parties that are never online at the same time, slowing PCS healing further.
- To assess claim C4, consider figure `fig6b_opp_vs_nonopp_unikem.png` which shows cumulative probability distributions of $|VulM|$ for the UniKEM and Opp-UniKEM protocol with different bandwidth limits. In the lowest bandwidth setting - a 32 byte limit per message - the opportunistic version of the protocol is clearly superior. This advantage persists but shrinks at the 128 byte limit, and disappears at the 512 byte limit.
- To assess claim C5, consider figure `fig7_vulnset_distributions.png` which plots cumulative probability distributions for all three opportunistic protocols under the four main messaging behaviors. It provides separate plots for compromise of user A and user B, since the opportunistic protocols have varying degrees of asymmetry. As claimed, the Opp-RKEM and Opp-UniKEM protocols perform similarly, and both outperform the Opp-BiKEM protocol for all messaging behaviors except the "disjoint desktop-desktop" model.

(E2): *[Appendix charts] [1 human-minute + 1 compute-minute + 6MB disk]: execute all protocols for all messaging behaviors and produce the charts in the paper and produce the charts in the appendix.*

Execution: From the main directory, run the following commands:

```
cd charts/
./appendix_tests.sh
```

Results: The figures from the appendix will be generated in the directory `charts/appendix_charts/`.

A.5 Notes on Reusability

While the scripts provided with this artifact are tailored to reproduce the charts in the paper, the underlying simulation code is designed for further experimentation. It is designed so that it is simple for a developer to implement and evaluate a new protocol or to add new messaging behaviors for evaluation of existing protocols - or both.

Adding a new protocol: To add a new protocol, implement the `MessagingScka` trait and the `ChunkedCKAVulnerability` trait for your protocol. Optionally, you can just implement the `Scka` trait for your protocol and use the `RkemMessagingScka` trait to get an implementation of the `MessagingScka` trait.

Adding a new messaging behavior: To add a new messaging behavior, implement the `MessagingBehavior` trait, which has a single function that returns a list of commands to be executed by the protocol participants at each time step.

Running simulations with a new protocol or messaging behavior: Once a developer has implemented a new protocol or messaging behavior, a simulation can be run as follows:

```
type Cka = MyCkaProtocol;
type MessagingBehavior = MyMessagingBehavior;
let mut mb = MyMessagingBehavior::new(...);
let numsteps = 100000usize;
let hist = orchestrator
    ::controlled_messaging_healing_test::<
    Cka,
    MessagingBehavior,
>(&mut mb, numsteps)
```

Looking through the code in `orchestrator.rs` will show other tests and measurements that can be performed, many of which are useful for testing the correctness of a protocol implementation. Look at the unit tests for existing protocols for examples.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at `https://secartifacts.github.io/usenixsec2025/`.