ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE | ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL | ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

USENIX'23 Artifact Appendix: Multi-Factor Key Derivation Function (MFKDF) for Fast, Flexible, Secure, & Practical Key Management

Vivek Nair
UC Berkeley
vcn@berkeley.edu

Dawn Song
UC Berkeley
dawnsong@berkeley.edu

A Artifact Appendix

MFKDF

Multi-Factor Key Derivation Function (MFKDF)
mfkdf.com | pbkdf2.com

A.1 Abstract

We present a JavaScript library implementing all of the factors and methods associated with the Multi-Factor Key Derivation Function (MFKDF) proposal. The library covers all proposed features of MFKDF, including threshold MFKDF, policy-based MFKDF, entropy measurement, authentication, and factor constructions for HOTP, TOTP, HMAC-SHA1, OOBA, and more. In separate repositories, we also include centralized and decentralized proof-of-concept web applications, along with a browser-based benchmarking suite. These repositories together contain about 100,000 lines of JavaScript code.

To aid evaluation, we have detailed documentation that includes usage examples for every supported method, as well as a series of in-depth tutorials. We have also included a unit testing suite with 100.0% code coverage, the results of which can be viewed online. We have compiled and hosted the demo applications and benchmarks for easy online access, and have included video tutorials explaining how to use them.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Our artifact is non-destructive, but caution should be taken when using the proof of concept applications, as they are intended for demonstration purposes only and have not been audited for security vulnerabilities. When evaluating the demo applications, evaluators should not use credentials that they have used or intend to use for any other application. Nothing of value should be stored in the ETH demo wallet at this time. We also recommend that any parties wishing to remain anonymous use a fictitious name and disposable email address.

A.2.2 How to access

MFKDF Library

- GitHub Repository (Stable): `https://github.com/multifactor/MFKDF/tree/1427224a709b77312b1b03cfa79ebed7bed316ea`
- Website: `https://mfkdf.com` (or `https://pbkdf2.com`)
- Documentation: `https://mfkdf.com/docs`
- Unit Testing Results: `https://mfkdf.com/tests`
- Code Coverage Report: `https://mfkdf.com/coverage`

Centralized Demo

- GitHub Repository (Stable): `https://github.com/multifactor/mfkdf-application-demo/tree/37ca96c58c050e460e6a3d4d09896eae06ed2720`
- Live Demo: `https://demo.mfkdf.com`
- Video: `https://youtube.com/watch?v=cB44BMGnFIs`

Decentralized Demo

- GitHub Repository (Stable): `https://github.com/multifactor/mfkdf-wallet-demo/tree/1fbc67d2b7505b2185a8ca3ce9ba163e22ae29ab`
- Live Demo: `https://wallet.mfkdf.com`
- Video: `https://youtube.com/watch?v=u3eUsPnv7K8`

Benchmarking

- GitHub Repository (Stable): `https://github.com/multifactor/mfkdf-benchmark/tree/72b81f89818b7b68313e05f17764aadb38fb99e0`
- Live Demo: `https://benchmark.mfkdf.com`

A.2.3 Hardware dependencies

Any system with a stable internet connection and capable of running JavaScript code should be able to run our demo applications and benchmarks. For consistency, the device we used for benchmarking has an AMD Ryzen 9 5950X CPU, NVIDIA GeForce RTX 3090 GPU, and 128GB of DDR4 RAM, although the benchmarking results almost exclusively depend on single-core CPU performance in this case. For the centralized demo, an iOS or Android mobile device capable of installing the Google Authenticator application is required. The decentralized demo application can use (but doesn't require) a YubiKey device supporting HMAC-SHA1.

A.2.4 Software dependencies

We expect that any device with a relatively modern web browser (HTML5/ES6) will be able to run our benchmarks and demo applications. Our evaluations were performed in Chrome Browser v103.0.5060.114 on Windows 10 v21H2.

If manually building any of the packages, each repository contains a `package.json` file with the names and versions of all dependencies. Node.js and NPM are required to build and test each package; we used Node.js v16.15.0 and NPM v8.4.0. Running `npm install` in the root directory of each repository should automatically install all of the dependencies.

A.2.5 Benchmarks

A self-contained benchmark that can run in any modern web browser (usually in less than a minute) is available at `https://benchmark.mfkdf.com`. The relevant source code is visible in index.html. No external data is required.

A.3 Set-up

A.3.1 Installation

1. Download and install the latest version of Google Chrome from `https://www.google.com/chrome`.
2. Download and install the latest LTS version of Node.js (including NPM) from `https://nodejs.org/en`.
3. On a mobile device, download and install the latest version of Google Authenticator for iOS or Android.
4. Clone our main GitHub repository by running `git clone http://github.com/multifactor/MFKDF`
5. Open the root directory of the repository (`cd MFKDF`), and then run `npm install` to download all dependencies.
6. Repeat steps 4 and 5 for each of the demo repositories if you wish to build them instead of using the hosted versions.

A.3.2 Basic Test

If the repository and dependencies have been installed correctly, running `npm run build` should successfully compile the package (you should see a message like `webpack X.X.X compiled with X warnings in X ms`).

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The MFKDF algorithms and constructions of §4–§9, as summarized by the pseudocode given in §A, can be used to produce a fully-functional MFKDF implementation satisfying the stated definitions and security goals of §3. This can be verified by experiment (E1), which runs a test suite enforcing the specification of the paper.

(C2): MFKDF has a low computational overhead over PBKDFs in a typical web browser. This can be verified by experiment (E2), which runs browser-based benchmarks for the setup and derive functions of a standard 3-of-3 MFKDF setup, a 2-of-3 threshold MFKDF setup, and all supported authentication factor constructions. The results can be used to verify the performance claims of §11 of the paper, particularly figures 7 and 8.

(C3): MFKDF can be used in place of PBKDFs in common centralized applications like password managers. Such applications can authenticate using derived keys (see §7), enforce arbitrarily specific derivation policies (see §9), and are fully backward-compatible with existing popular authentication factors like TOTP (see §5). This can be verified by experiment (E3), which evaluates a functional MFKDF-based password management application using the standard Google Authenticator mobile app, and can in turn can be used to confirm §10.1 of the paper.

(C4): MFKDF can be used to enable new applications in situations where PBKDFs would not be used, such as in fully-decentralized applications. The public parameters (α) can be stored openly, such as on a public blockchain. This can be verified by experiment (E4), which evaluates a decentralized Ethereum and ERC20 wallet based on MFKDF and stores parameters using IPFS and IPNS, per the description of §10.2 of the paper.

A.4.2 Experiments

(E1): *[Unit Tests] [5 human-minutes + 5 compute-minutes]: Run the included unit testing suite to verify that all tests are passing with 100.0% code coverage.*
Preparation: Clone the main MFKDF repository and install the dependencies as described in §A.3.1.
Execution: Simply run `npm test` to simultaneously generate testing and code coverage results. The tests assert the library's compliance with the specifications of the paper; you can browse the `test` directory to verify.
Results: After about 2 minutes, `337 passing` should be displayed with no tests failing. Below that, the code coverage report should show 100% for all files.

(E2): *[Benchmark] [5 human-minutes + 1 compute-minute]: Run our browser-based benchmark to replicate the main performance evaluation described in the paper.*
Preparation: Visit our hosted benchmarking page at `https://benchmark.mfkdf.com` (recommended), or clone the benchmarking repository and read `README.md`.
Execution: Simply click "run now" to benchmark MFKDF, threshold MFKDF, and all supported factors. You can browse the source code to verify its validity.
Results: After about 1 minute, a results table should be displayed that roughly matches Fig. 7 and Fig. 8 of the paper. The scripts for generating these figures using the benchmarking output are included in the `figs` directory.

(E3): *[Centralized Demo] [30 human-minutes]: Run our browser-based centralized proof-of-concept to verify the compatibility results described in §10.1 of the paper.*
Preparation: Visit our hosted application demo at `https://demo.mfkdf.com` (recommended), or install the repository manually per §A.3.1 and run `npm build`.
Execution: Create an account on the demo application using false information and the Google Authenticator app on your mobile device. Use an anonymous, but valid, email address that can receive mail. Store at least one fictitious password, then log out. To store a password, type a site name (e.g., google.com), then click on the correct option from the dropdown, and type a username and password before clicking save. Log back in using your master password and TOTP, or try the various recovery options, and verify that you can still access the stored password. Our demo video shows the intended usage.
Results: Confirm that the application behaves as described in the paper. Specifically, it should be fully backward-compatible with the Google Authenticator mobile app, and that incorrect login factors can't be used to successfully access encrypted information. You can verify that the source code uses MFKDF as described.

(E4): *[Decentralized Demo] [30 human-minutes]: Run our browser-based decentralized proof-of-concept to verify the functionality described in §10.2 of the paper.*
Preparation: Visit our hosted application demo at `https://wallet.mfkdf.com` (recommended), or install the repository per §A.3.1 and run `npm build`.
Execution: Use false credentials to create a wallet for the Ethereum mainnet or Ropsten testnet. Note the username and recovery code. Optionally, you may transfer nominal funds to the wallet address using a free Ropsten faucet. Sign out, then log back in using your password and recovery code, and verify that you can still access the funds. Our demo video shows the intended usage.
Results: Confirm that the wallet behaves as described in the paper. Verify that incorrect login factors can't be used to successfully access the wallet. You can check that the source code uses MFKDF as described.
***Note:** In the time between submission and publication, the Ethereum merge caused the Ropsten testnet to shut down. Our new and improved demo, using the Sepolia testnet, can be found at `https://ciao.mfkdf.com`.

A.5 Archive of Repositories

The version of each repository evaluated by the USENIX AEC has been tagged with "usenix-ae" on GitHub. A copy of each of these repositories has also been uploaded to Zenodo in their evaluated state for historical preservation:

`https://doi.org/10.5281/zenodo.7859226`

A.6 Notes on Reusability

The MFKDF JavaScript library was built with the express goal of being flexible and easy to deploy in a wide variety of new or existing applications. Its creative commons license puts almost no restrictions on non-commercial use.

We suggest that interested parties get started by visiting mfkdf.com to learn more, reading our tutorial series, browsing our detailed documentation, and trying our demos.

The MFKDF library is flexible, modular, and easy to extend with new features and factor types. If you are interested in contributing, please read our contributing guide.

The benchmarking code and two demo applications are all offered under the MIT license, and can be modified and redistributed essentially without restriction.

A.7 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at `https://secartifacts.github.io/usenixsec2023/`.