Predicting performance for post-quantum encrypted-file systems

Daniel J. Bernstein[1,2]

[1] Department of Computer Science, University of Illinois at Chicago, USA
[2] Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
djb@cr.yp.to

Abstract. Public-key cryptography is widely deployed for encrypting stored files. This paper uses microbenchmarks and purchase costs to predict the performance of various post-quantum KEMs in this application, in particular concluding that Classic McEliece is (1) the most efficient option and (2) easily affordable.

Keywords: post-quantum cryptography, encryption, performance

1 Introduction

Windows Pro, Enterprise, and Education include an "Encrypting File System". Microsoft describes EFS in [14] as "the built-in file encryption tool for Windows file systems", and explains that each file stored under EFS is automatically encrypted to the public key of the user authorized to access the file:

> EFS enables transparent encryption and decryption of files by using advanced, standard cryptographic algorithms. Any individual or program that doesn't possess the appropriate cryptographic key cannot read the encrypted data. Encrypted files can be protected even from those who gain physical possession of the computer that the files reside on. ... EFS encryption doesn't occur at the application level but rather at the file-system level; therefore, the encryption and decryption process is transparent to the user and to the application. ... File encryption uses a symmetric key, which is then itself encrypted with the public key of a public key encryption pair. The related private key must be available in order for the file to be decrypted. This key pair is bound to a user identity and made available to the user who has possession of the user ID and password.

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Excellence Strategy of the German Federal and State Governments—EXC 2092 CASA—390781972 "Cyber Security in the Age of Large-Scale Adversaries"; by the U.S. National Science Foundation under grant 2037867; and by the Taiwan's Executive Yuan Data Safety and Talent Cultivation Project (AS-KPQ-109-DSTCP). "Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation" (or other funding agencies). Permanent ID of this document: 27c1d851ac2f907d715a0bbad5fa60623088bc54. Date: 2023.12.06.

Windows also supports full-disk encryption ("BitLocker"). Microsoft's BitLocker FAQ [21] explains that per-file encryption and full-disk encryption provide different security features:

> Encrypting File System (EFS) can be used to encrypt files on a BitLocker-protected drive. BitLocker helps protect the entire operating system drive against offline attacks, whereas EFS can provide additional user-based file level encryption for security separation between multiple users of the same computer. EFS can also be used in Windows to encrypt files on other drives that aren't encrypted by BitLocker.

There are many other examples of per-file public-key encryption. Apple's iPhone, despite its reputation as a single-user device protected by full-disk encryption, also automatically creates ciphertexts to the user's Curve25519 public key, for example for backups and to be able to encrypt incoming messages while the device is locked. See [5, pages 80 and 82]. For Linux tools that automatically encrypt files to a per-user GPG public key, see, e.g., [15], [19], and [23] for incoming mail messages, or [24] for incoming HTTP uploads.

To protect against an attacker with a quantum computer, one can and should upgrade encrypted-file systems from pre-quantum public-key encryption to post-quantum public-key encryption: for example, choose any post-quantum KEM, generate a KEM ciphertext to the user's KEM public key for each encrypted file, and use the resulting KEM session key to encrypt the file, or to encrypt another (perhaps longer-lasting) symmetric key used in turn to encrypt the file. "Hybrid"/"double encryption" approaches, retaining the existing layer of public-key encryption while adding a new layer of post-quantum public-key encryption, are also straightforwardly applicable (and recommended) here: e.g., encrypt with the old layer and then the new, or vice versa, or hash session keys and ciphertexts obtained from the old and new layers to obtain a hybrid session key used to encrypt the file.

1.1. Contributions of this paper. This paper asks what the performance consequences are of upgrading encrypted-file systems to use post-quantum cryptography. This paper uses microbenchmarks from eBACS [12] to predict the answer for post-quantum KEMs currently under consideration by NIST and/or ISO: BIKE, Classic McEliece, FrodoKEM, and Kyber. One patented KEM, HQC, has not submitted up-to-date software to eBACS and is omitted.

This comparison is analogous to, e.g., [17] comparing some post-quantum KEMs according to "compactness" and "the round-trip time of ephemeral key exchange". The latter is computed in [17] as the sum of microbenchmarks for key generation, encapsulation, and decapsulation, disregarding KEM-independent overheads such as end-to-end communication latency. Two differences between this paper's comparison and the comparison in [17] are as follows:

- Beyond separately measuring storage space, communication, and CPU time, this paper uses purchase costs as a way to put all of these on the same scale, whereas [17] treats space and time as independent, not even accounting for the effect of "compactness" upon "round-trip time".

- An encrypted-file system reuses a user's public key for all of the files encrypted to that user, whereas [17] considers a protocol that generates a new key for every ciphertext. All of the KEMs considered in this paper are designed for IND-CCA2 security, allowing keys to be reused for many ciphertexts.

These differences change which KEMs score best. Consider, in particular, Classic McEliece, the post-quantum encryption system with by far the strongest security track record. This system is often claimed to be non-competitive in performance because it has much larger per-key costs than alternatives: e.g., 1047319 bytes for a `mceliece6960119` public key, compared to 1568 bytes for a `kyber1024` public key. However, Classic McEliece has much smaller per-ciphertext costs than alternatives: e.g., 194 bytes for a `mceliece6960119` ciphertext, compared to 1568 bytes for a `kyber1024` ciphertext. Classic McEliece ends up being the most efficient way to encrypt many files to one key.

1.2. Variables. The predictions throughout this paper are in terms of the following variables. There are U users of the file system; e.g., $U = 1$ for a single-user file system. There are E files encrypted to an average user, for a total of UE encrypted files. There are D file-decryption operations by an average user, for a total of UD decrypted files. There are S files encrypted to an average user and still stored, for a total of US stored encrypted files. There is then a choice of KEM.

This paper takes four concrete examples to illustrate the impact of variations in E, D, and S:

- $(E, D, S) = (100000, 200000, 50000)$. This models a user receiving and storing 100 messages per work day for 5 years in an encrypted-file system, assuming 200 work days per year, while decrypting each message twice on average. S is half as large as E since at an average moment only half of the messages have arrived.
- $(E, D, S) = (200000, 400000, 100000)$. This models a more active user.
- $(E, D, S) = (200000, 200000, 100000)$. Same, except that the user decrypts each message only once on average.
- $(E, D, S) = (200000, 200000, 2000)$. Same, except that the user has a message-deletion policy and deletes an average message in just 10 work days.

This paper presumes that servers will be replaced with new, more cost-effective servers after 5 years, outweighing any increase in the number of messages, so this paper focuses on costs in the first 5 years.

Here is a sanity check on the 100-message-per-day number: [26] indicates that there were 333.2 billion email messages sent worldwide each day in 2022, the equivalent of 600 billion messages per work day. Of course, email messages are not necessarily stored in encrypted-file systems (even if they should be for security); in the opposite direction, email messages are only one example of files that can be stored in encrypted-file systems. It would be interesting to collect real-world data regarding the variables (E, D, S).

	$S = 2000$	$S = 50000$	$S = 100000$
mceliece348864f	459612	5067612	9867612
mceliece460896f	849768	8337768	16137768
mceliece6960119f	1449267	10761267	20461267
mceliece6688128f	1474924	11458924	21858924
mceliece8192128f	1787944	11771944	22171944
kyber512	1538432	38402432	76802432
kyber768	2179584	54403584	108803584
kyber1024	3140736	78404736	156804736
bikel1	3152764	78656764	157306764
bikel3	6243188	155763188	311513188
frodokem640aes	19469504	486029504	972029504
frodokem640shake	19469504	486029504	972029504
frodokem976aes	31534928	787246928	1574446928
frodokem976shake	31534928	787246928	1574446928
frodokem1344aes	43328608	1081664608	2163264608
frodokem1344shake	43328608	1081664608	2163264608

Table 2.1.1. Number of bytes stored by each KEM per user of an encrypted-file system. S is the number of files stored per user. Rows are sorted by the last column. Columns are sorted by S.

2 Space

This section predicts the space consumption of various KEMs in the context of encrypted-file systems. This section also reviews the purchase cost of storage.

2.1. Bytes stored. Each user incurs KEM-specific storage for a KEM public key and a KEM private key, so there are U public keys and U private keys. Each stored file incurs KEM-specific storage for a KEM ciphertext, so there are US ciphertexts. The KEM-specific storage consumed *per user* is thus 1 public key plus 1 private key plus S ciphertexts. Table 2.1.1 reports this amount of storage for the examples of S listed in Section 1.2. All tables in this paper are produced by Python scripts provided as PDF attachments to this paper.

2.2. The purchase cost of storage per byte. A typical 20TB hard drive today, the Western Digital WD201KFGX, is available for under $400. The data sheet for this drive [**30**] indicates average power consumption of 6.9 watts during read/write and 1.6 watts on standby. If the drive runs continuously for its 5-year warranty period then it consumes between 70 and 300 kilowatt-hours depending on how frequent the read/write operations are. A typical power supply is over 90% efficient, and then purchasing this energy at a typical price of 10 cents per kilowatt-hour adds between $7 and $33, making little difference next to the $400 purchase price of the drive itself.

During the same 5-year period, the drive provides 100 terabyte-years of storage; i.e., one terabyte-year of storage costs about $4. In other words, storing a byte for a year costs about 2^{-38} dollars.

Costs vary depending on the storage technology. For example, SSDs are often chosen over hard drives because they provide data access with lower latency and higher throughput; but a 4TB SSD currently costs about $200, more than twice as expensive per byte as the hard drive. Also, a typical SSD warranty is limited to 5 years *or* a few petabytes written, so applications that rewrite all their data every day (e.g., continuously recording video feeds and retaining them for 24 hours) will end up replacing SSDs roughly every year, further increasing the cost per byte-year of storage.

Another reason for higher costs per byte-year is overprovisioning. For example, a user purchasing much more storage for a laptop than currently needed, so as to leave room for future growth, ends up paying more per byte-year of storage.

3 Communication

This section predicts the amount of data read and written by various KEMs in the context of encrypted-file systems. This section also reviews the purchase cost of the relevant communication mechanisms.

3.1. Bytes written and read. Each user incurs the cost of writing a KEM public key and a KEM private key, so there are U public keys written and U private keys written. This paper assumes that caching of user keys in RAM is sufficiently effective that the cost of reading keys can be ignored.

Each encrypted file incurs the cost of writing a KEM ciphertext, so there are UE ciphertexts written. Each file-decryption operation incurs the cost of reading a KEM ciphertext, so there are UD ciphertexts read.

The KEM-specific reading and writing *per user* is thus 1 public key plus 1 private key plus $E + D$ ciphertexts. Table 3.1.1 computes this number of bytes for the examples of (E, D) listed in Section 1.2.

3.2. The purchase cost of communication per byte. The drive described in Section 2.2 has an internal transfer rate of 268 MB/second, according to [**30**]. Reading or writing data adds 5.3 watts compared to standby and, perhaps more relevant to marginal cost accounting, 3.1 watts compared to idle. Either way, the cost is roughly 2^{-51} dollars per byte at 10 cents per kWh.

Remote file systems are more expensive. For example, storing data in the cloud incurs costs to transmit data to and from the cloud, although it has the benefit of reducing the need for overprovisioning.

One way to estimate the purchase price of a byte of Internet communication is to look at advertisements of, e.g., $70/month for a gigabit (roughly 100MB/second) connection, suggesting a cost around 2^{-42} dollars per byte. There are, however, widespread reports of these connections being limited to a few terabytes per month, roughly 2^{-36} dollars per byte.

This paper uses an intermediate estimate obtained as follows. A 2018 estimate [**6**] was that Internet data transmission costs "0.06 kWh/GB", with a historical trend of dropping by a factor 2 every 2 years. At 10 cents per kWh, this is 0.006 dollars per GB in 2018, or projected 2^{-40} dollars per byte in 2024.

	$E = 100000$ $D = 200000$	$E = 200000$ $D = 200000$	$E = 200000$ $D = 400000$
mceliece348864f	29067612	38667612	57867612
mceliece460896f	47337768	62937768	94137768
mceliece6960119f	59261267	78661267	117461267
mceliece6688128f	63458924	84258924	125858924
mceliece8192128f	63771944	84571944	126171944
kyber512	230402432	307202432	460802432
kyber768	326403584	435203584	652803584
kyber1024	470404736	627204736	940804736
bikel1	471906764	629206764	943806764
bikel3	934513188	1246013188	1869013188
frodokem640aes	2916029504	3888029504	5832029504
frodokem640shake	2916029504	3888029504	5832029504
frodokem976aes	4723246928	6297646928	9446446928
frodokem976shake	4723246928	6297646928	9446446928
frodokem1344aes	6489664608	8652864608	12979264608
frodokem1344shake	6489664608	8652864608	12979264608

Table 3.1.1. Number of bytes read+written by each KEM per user of an encrypted-file system. E is the number of files encrypted per user. D is the number of files decrypted per user. Rows are sorted by the last column. Columns are sorted by (E, D).

Recall from Section 2.2 the estimate of about 2^{-38} dollars per byte-year stored. If data is stored for only 10 days, about 2^{-5} years, then this storage costs 2^{-43} dollars per byte, so it is outweighed by 2^{-40} dollars per byte to upload to the cloud (plus the same for each download); communication costs for cloud storage end up dominant for, e.g., $(E, D, S) = (200000, 200000, 2000)$. If data is instead stored for years then a few uploads and downloads are outweighed by the data-storage costs: the data-storage costs end up dominant for, e.g., $(E, D, S) = (200000, 400000, 100000)$.

4 CPU time

This section predicts the number of CPU cycles used by various KEMs in the context of encrypted-file systems. This section also reviews the purchase cost of CPU cycles.

4.1. Cycles used. Each user generates a KEM public key and a KEM private key, so there are U key-generation operations. Each encrypted file incurs a KEM-specific encapsulation operation, so there are UE encapsulation operations. Each decryption of a file incurs a KEM-specific decapsulation operation, so there are UD encapsulation operations.

The KEM cycles *per user* are thus 1 key generation plus E encapsulations plus D decapsulations. Table 4.1.1 computes the number of Skylake cycles for

	$E = 100000$ $D = 200000$	$E = 200000$ $D = 200000$	$E = 200000$ $D = 400000$
kyber512	9160223055	12738623055	18320423055
kyber768	13803039812	19140439812	27606039812
kyber1024	19515254565	26862854565	39030454565
mceliece348864f	25341940075	28417140075	50655140075
mceliece460896f	54924488505	61773088505	109756488505
mceliece6960119f	65542051009	78232651009	130902451009
mceliece6688128f	69839600596	81593300596	139453300596
mceliece8192128f	72425959005	86494859005	144619259005
bikel1	324313986627	335022786627	648627386627
frodokem640aes	545458931360	730642231360	1090916431360
bikel3	1001268224846	1026646824846	2002534824846
frodokem976aes	1048656438075	1414876938075	2097309938075
frodokem640shake	1250679172307	1669926472307	2501354472307
frodokem1344aes	1729058990308	2318073590308	3458112990308
frodokem976shake	2657322821805	3556652121805	5314637321805
frodokem1344shake	4608595220865	6163803420865	9217175820865

Table 4.1.1. Number of Skylake cycles used by each KEM per user of an encrypted-file system. E is the number of files encrypted per user. D is the number of files decrypted per user. Rows are sorted by the last column. Columns are sorted by (E, D).

these operations for the examples of (E, D) listed in Section 1.2, starting from the median cycle counts from [12].

4.2. The purchase cost of CPU time per cycle. A Dell PowerEdge T440 server with two Intel Xeon Silver 4216 CPUs, 16GB of ECC RAM, and a 5-year warranty costs $3787 today from https://dell.com. This is not the lowest-cost processing currently available, but it suffices for purposes of this paper.

Each of the two CPUs has 16 cores and a base frequency of 2.1GHz; this paper assumes that Turbo Boost is disabled. The CPU cores together carry out $2^{63.2}$ cycles in 5 years. The CPU microarchitecture is Cascade Lake, which is essentially Skylake plus AVX-512; beware that Table 4.1.1 does not use AVX-512 and is presumably overestimating costs on these CPUs.

The server has a 495W power supply, but can be reasonably estimated to draw under 300W from the wall since each CPU has a thermal design power of 100W. Running continuously for 5 years (i.e., 43.83 kilo-hours) then costs under $1315 at 10 cents per kWh.

Overall a CPU cycle on these CPUs costs about 2^{-51} dollars. Recall from Section 3.2 that reading a byte from a local hard drive also costs roughly 2^{-51} dollars, but reading a byte from remote storage costs roughly 2^{-40} dollars; and recall from Section 2.2 that a byte-year of storage costs about 2^{-38} dollars.

	$E = 100000$ $D = 200000$ $S = 50000$	$E = 200000$ $D = 200000$ $S = 2000$	$E = 200000$ $D = 200000$ $S = 100000$	$E = 200000$ $D = 400000$ $S = 100000$
`mceliece348864f`	0.00008501	0.00001933	0.00015623	0.00016611
`mceliece460896f`	0.00014574	0.00003983	0.00026230	0.00028362
`mceliece6960119f`	0.00018573	0.00005587	0.00033253	0.00035594
`mceliece6688128f`	0.00019779	0.00005774	0.00035436	0.00038007
`mceliece8192128f`	0.00020350	0.00006447	0.00036109	0.00038692
`kyber512`	0.00056300	0.00002818	0.00112342	0.00112596
`kyber768`	0.00079795	0.00004041	0.00159199	0.00159585
`kyber1024`	0.00114981	0.00005791	0.00229402	0.00229956
`bikel1`	0.00128884	0.00019494	0.00243817	0.00257758
`bikel3`	0.00271172	0.00054733	0.00498959	0.00542325
`frodokem640aes`	0.00731619	0.00060952	0.01447109	0.01463195
`frodokem640shake`	0.00762937	0.00102664	0.01488821	0.01525831
`frodokem976aes`	0.01192375	0.00109002	0.02354235	0.02384681
`frodokem976shake`	0.01263814	0.00204116	0.02449349	0.02527559
`frodokem1344aes`	0.01651103	0.00166379	0.03251292	0.03302112
`frodokem1344shake`	0.01778980	0.00337164	0.03422076	0.03557866

Table 5.1.1. Estimated dollar costs for each KEM per user of an encrypted-file system in the local-storage scenario. E is the number of files encrypted per user. D is the number of files decrypted per user. S is the number of files stored per user. Rows are sorted by the last column. Columns are sorted by (E, D, S).

5 Total costs

This section predicts the total dollar costs per user of various KEMs in the context of encrypted-file systems. These predictions are split into two scenarios covered in Section 3.2: data being stored locally on a hard drive, and data being stored remotely in the cloud.

5.1. The local-storage scenario. Table 5.1.1 makes predictions for the local-storage scenario. The cost of 5 years of storing a byte is estimated as 2^{-36} dollars; the cost of reading or writing a byte is estimated as 2^{-51} dollars; the cost of a cycle is estimated as 2^{-51} dollars. These estimates come from Sections 2.2, 3.2, and 4.2.

The minimum per-user cost in Table 5.1.1 is from `mceliece348864f` for $(E, D, S) = (200000, 200000, 2000)$; recall that this is the setting of 200 files per day being encrypted to a user (and then decrypted) but files being deleted after 10 days. Here is a spot-check on this table entry:

- `mceliece348864f` has a 96-byte ciphertext for each of the 2000 stored files, occupying 192000 bytes of storage. `mceliece348864f` also has a 261120-byte public key and a 6492-byte private key, so the total storage is 459612 bytes. Multiplying by the estimated 2^{-36} dollars per byte gives 6.688 microdollars.

- `mceliece348864f` also writes 96-byte ciphertexts for 200000 encryption operations, reads 96-byte ciphertexts for 200000 decryption operations, and writes a public key and a private key, for a total of 38667612 bytes read and written. Multiplying by the estimated 2^{-51} dollars per byte gives 0.017 microdollars. (In Section 5.2, this 2^{-51} changes to 2^{-40}, increasing this cost to 35.168 microdollars.)
- The median cycle counts for `mceliece348864f` from [**12**] are 28740075 cycles for keygen, 30752 cycles for enc, and 111190 cycles for dec, so 1 keygen, 200000 enc, and 200000 dec consume a total of 28.417 billion cycles. Multiplying 28.417 billion cycles by the estimated 2^{-51} dollars per cycle gives 12.620 microdollars.

The total matches the 19.33 microdollars (0.00001933) in Table 5.1.1.

In the same $(E, D, S) = (200000, 200000, 2000)$ setting, `kyber512` is 28.18 microdollars per user; `mceliece460896f` is 39.83 microdollars per user; `kyber768` is 40.41 microdollars per user; `mceliece6960119f` is 55.87 microdollars per user; `kyber1024` is 57.91 microdollars per user. Note that precise comparisons are uncertain since the numbers come from rough estimates; the reason for reporting extra digits is to support spot-checks.

For the other (E, D, S) settings in Table 5.1.1, the KEM costs are considerably higher and are dominated by ciphertext storage. This also makes Kyber strikingly less efficient than Classic McEliece.

5.2. The cloud-storage scenario. Table 5.2.1 makes predictions for the cloud-storage scenario, with reading or writing a byte estimated as 2^{-40} dollars. The cost of 5 years of storing a byte is estimated as 2^{-36} dollars; the cost of reading or writing a byte is estimated as 2^{-40} dollars (this is what is different from Section 5.1); the cost of a cycle is estimated as 2^{-51} dollars. These estimates come from Sections 2.2, 3.2, and 4.2.

Classic McEliece is the clear winner across all of the (E, D, S) settings in Table 5.2.1. The closest competition is for $(E, D, S) = (200000, 200000, 2000)$, where `mceliece348864f` costs 54.48 microdollars per user, `mceliece6960119f` costs 127.37 microdollars per user, `kyber512` costs 307.44 microdollars per user, and `kyber1024` costs 628.07 microdollars per user.

6 Costs in context

This section compares the costs of post-quantum encrypted-file systems to other file-system costs and to costs of other applications of post-quantum cryptography.

6.1. Other file-system costs. The numbers in Tables 5.1.1 and 5.2.1 for `mceliece6960119f`, which is designed for long-term security, are under 1/2000 of a dollar per user, suggesting that deployment in this application is very easily affordable. However, costs could be higher than this, for example because more expensive storage technologies are being used (see Section 2.2) or because many more files are being stored.

	$E = 100000$ $D = 200000$ $S = 50000$	$E = 200000$ $D = 200000$ $S = 2000$	$E = 200000$ $D = 200000$ $S = 100000$	$E = 200000$ $D = 400000$ $S = 100000$
`mceliece348864f`	0.00011143	0.00005448	0.00019138	0.00021872
`mceliece460896f`	0.00018878	0.00009704	0.00031951	0.00036919
`mceliece6960119f`	0.00023960	0.00012737	0.00040403	0.00046271
`mceliece6688128f`	0.00025548	0.00013433	0.00043096	0.00049449
`mceliece8192128f`	0.00026147	0.00014135	0.00043797	0.00050162
`kyber512`	0.00077245	0.00030744	0.00140268	0.00154486
`kyber768`	0.00109467	0.00043603	0.00198762	0.00218928
`kyber1024`	0.00157744	0.00062807	0.00286418	0.00315480
`bikel1`	0.00171783	0.00076692	0.00301015	0.00343555
`bikel3`	0.00356124	0.00168002	0.00612228	0.00712228
`frodokem640aes`	0.00996701	0.00414393	0.01800550	0.01993356
`frodokem640shake`	0.01028019	0.00456106	0.01842263	0.02055992
`frodokem976aes`	0.01621742	0.00681490	0.02926723	0.03243410
`frodokem976shake`	0.01693181	0.00776604	0.03021837	0.03386288
`frodokem1344aes`	0.02241046	0.00952968	0.04037881	0.04481993
`frodokem1344shake`	0.02368923	0.01123753	0.04208666	0.04737746

Table 5.2.1. Estimated dollar costs for each KEM per user of an encrypted-file system in the cloud-storage scenario. E is the number of files encrypted per user. D is the number of files decrypted per user. S is the number of files stored per user. Rows are sorted by the last column. Columns are sorted by (E, D, S).

A different way to build confidence in the affordability of post-quantum encrypted-file systems is to compare the costs of post-quantum cryptography to other file-system costs. For example, measurements of the distribution of file sizes (see, e.g., [27] and [16]) indicate that the average file size has grown past a megabyte, making it implausible that there can be an issue with adding a relatively small KEM ciphertext to each file.

A counterargument is as follows. Both [27] and [16] show a large spread of file sizes, with the average file size differing across use cases. For example, [16] observes work by "IT staff (e.g., programmer, systems administrator)" storing much smaller files than other use cases, and explains this via a previous study of the prevalence of "plain text files", "documents", and "media" across different use cases.

It would be interesting to collect statistics on the sizes of files stored on encrypted-file systems: for example, how do incoming messages automatically encrypted by an iPhone compare in size to other files? It would also be interesting to collect statistics on how often files of different sizes are decrypted.

It is conceivable that there are use cases that store a huge number of 100-byte files in an encrypted-file system and that would object to adding 194 bytes per file for a `mceliece6960119f` ciphertext. Quantitatively, a dollar pays for 5-year storage of about a quarter billion of these ciphertexts; what happens if a user is storing many billions of files?

However, it is reasonably clear that such use cases are not common, since they would already incur larger overheads for unencrypted-file systems. For example, Windows NTFS allocates storage in units of "clusters", and NTFS "clusters" cannot be smaller than 512 bytes (see [29]), so every 100-byte file is wasting 412 bytes of disk space.

To summarize, the evidence indicates that upgrading encrypted-file systems to `mceliece6960119f` will be easily affordable for practically all use cases.

6.2. Other applications. The affordability of this upgrade to post-quantum cryptography sounds very different from a common narrative that post-quantum performance is a major issue. Consider, for example, NIST's round-3 report [3] commenting extensively on post-quantum performance, deferring a decision on whether to standardize Classic McEliece, and rejecting FrodoKEM outright:

> Classic McEliece was a finalist, but is not being standardized by NIST at this time. Although it is widely regarded as secure, NIST does not yet anticipate it being widely used because of its large public key size ... Frodo is clearly not an immediate drop-in general-purpose scheme.

One might try to explain the different conclusions regarding affordability by hypothesizing that

- encrypted-file systems are an unusual application because of their reuse of public keys, while
- *normal* applications generate a new public key for every ciphertext, involving thousands of times more data for Classic McEliece than just the ciphertexts.

But it is very easy to find literature on protocols relying on IND-CCA2 security to be able to reuse keys. Furthermore, long-term identity keys are used pervasively in protocol design, and can be long-term signature keys but—in any protocol where the signer is online—can also be long-term encryption keys, as in, e.g., [7, Section 3.2], [8], [9], [10, Section 8], and [25]. The available evidence indicates that post-quantum encryption is more efficient than post-quantum signatures. A full analysis of the efficiency of post-quantum cryptography certainly has to include long-term post-quantum encryption keys, contrary to the idea that each encryption key is used just once.

Even in situations where there *is* a new public key for every ciphertext, this merely doubles the amount of FrodoKEM data. Why is FrodoKEM "clearly not an immediate drop-in general-purpose scheme"? One might try to answer this question by formulating the following further hypotheses:

- Encrypted-file systems, at least for the limited sizes of (E, D, S) in Tables 5.1.1 and 5.2.1, are far below the overall volume of "general-purpose" usage of public-key cryptography, in particular for retrieving web pages.
- Quantitatively, even though those tables estimate 5 years of active use of an encrypted-file system as costing just pennies per user with FrodoKEM, users incur many more public-key operations for retrieving web pages, and

> with FrodoKEM or Classic McEliece those operations would add up to a *significant* cost.

But there does not appear to be any documentation quantifying the dollar cost here.

An earlier NIST report [2] stated that FrodoKEM in TLS key exchange would cost "around 20,000 bytes" plus "2 million cycles" for the server, and concluded that FrodoKEM does not have "acceptable performance in widely used applications overall". In response to the question of why this cost was not "acceptable", NIST wrote the following [22]:

> While it is not possible to speak for what every user of our standards would or wouldn't find "acceptable", there is a pretty large difference between the performance of Frodo on the one hand and Kyber, NTRU, and Saber on the other hand. We are therefore more confident that Kyber, NTRU, or Saber will be considered "acceptable" for most users than that Frodo will.

Quantifying costs in context produces different conclusions. The average web page has grown past 2 megabytes (see [18]), and a single TLS session can retrieve any number of web pages from a server. A web page can collect data from multiple servers, but the number of TCP connections per web page has dropped to 10 (see again [18]), so the number of TLS sessions involved is at most 10. In other words, the 20000 bytes for FrodoKEM cannot add more than 10% to overall web-page traffic, and if a user is retrieving at least 10 web pages per TLS session then 20000 bytes cannot add more than 1%.

NIST's calculation of 20000 bytes implicitly assumes that a key is transmitted for every ciphertext—which is another indication of the application not caring about key-distribution costs, given that those costs can obviously be reduced:

- A server that announces a new key every day can receive any number of ciphertexts to that key from any number of clients. Each client receives at most one key from the server per day—for example, spending at most 2^{-20} dollars per server per day for Classic McEliece keys.
- Keys also do not need to be sent from end to end: they can use lower-cost broadcast networks, for example using existing ISP-level caches to share key-retrieval costs across users. This can turn Classic McEliece into the lowest-cost KEM (see [4, Section 4] and [11, pages 16–18]), much as in the case of encrypted-file systems.
- Users are often on more expensive cellular networks, but they can download keys for frequently used servers in advance. An iPhone already automatically waits until it is on a wireless network before downloading software upgrades; it can do the same to reduce the cost of downloading new public keys. Servers can also distribute multiple keys in advance to simplify download scheduling, for example distributing the keys to be used for the next 7 days; this is compatible with erasing each key within a day of user data being sent to that key.

The absence of these features in TLS today is easily explained by (1) the history of TLS using small-key cryptosystems and (2) the lack of evidence that the key-distribution costs matter to the end user.

Recall the estimate from Section 3.2 of 2^{-40} dollars per byte for Internet communication. This corresponds to 50 million 20000-byte FrodoKEM sessions per dollar: i.e., spending a dollar on FrodoKEM traffic would pay for 50000 new FrodoKEM sessions per work day for 5 years. These are sizes for `frodokem640`, but `frodokem1344` is only about twice as large. It would be interesting to collect data on the number of TLS sessions initiated in a day by an average user's browser, the number of different TLS servers contacted, etc.

In general, given trends towards larger volumes of data, one would expect that the costs of post-quantum public-key encryption will ultimately be irrelevant compared to the costs of symmetric cryptography and non-cryptographic costs. This is not necessarily the situation today, but it is concerning to see unsubstantiated claims of unaffordability being used to discourage deployment of lower-risk cryptosystems.

References

[1] — (no editor), *Information: equity, diversity, inclusion, justice, and relevance—proceedings of the 84th ASIS&T annual meeting, ASIST 2021, Salt Lake City, UT, USA, October 30–November 2, 2021*, Proceedings of the Association for Information Science and Technology, 58, Wiley, 2021. See [16].

[2] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, *Status report on the second round of the NIST Post-Quantum Cryptography Standardization Process* (2020). NISTIR 8309. URL: https://csrc.nist.gov/publications/detail/nistir/8309/final. Citations in this document: §6.2.

[3] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, *Status report on the third round of the NIST Post-Quantum Cryptography Standardization Process* (2022). NISTIR 8413. URL: https://web.archive.org/web/20230824124130/https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413-upd1.pdf. Citations in this document: §6.2.

[4] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, Wen Wang, *Classic McEliece: conservative code-based cryptography: guide for implementors* (2022). URL: https://classic.mceliece.org/mceliece-impl-20221023.pdf. Citations in this document: §6.2.

[5] Apple, *Apple Platform Security* (2022). URL: https://web.archive.org/web/20231128180844/https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf. Citations in this document: §1.

[6] Joshua Aslan, Kieren Mayers, Jonathan G. Koomey, Chris France, *Electricity intensity of Internet data transmission: untangling the estimates*, Journal of Industrial Ecology **22** (2018), 785–798. URL: https://onlinelibrary.wiley.com/doi/10.1111/jiec.12630. Citations in this document: §3.2.

[7] Mihir Bellare, Ran Canetti, Hugo Krawczyk, *A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract)*, in STOC 1998 [**28**] (1998), 419–428. URL: https://eprint.iacr.org/1998/009. Citations in this document: §6.2.

[8] Daniel J. Bernstein, *DNSCurve: Usable security for DNS* (2009). URL: https://dnscurve.org. Citations in this document: §6.2.

[9] Daniel J. Bernstein, *The post-quantum Internet* (2016). URL: https://cr.yp.to/talks.html#2016.02.24. Citations in this document: §6.2.

[10] Daniel J. Bernstein, *D2.5: Internet: Integration* (2018). URL: https://www.pqcrypto.eu/deliverables/d2.5.pdf. Citations in this document: §6.2.

[11] Daniel J. Bernstein, *Migrating to the McEliece cryptosystem* (2023). URL: https://cr.yp.to/talks.html#2023.10.25. Citations in this document: §6.2.

[12] Daniel J. Bernstein, Tanja Lange (editors), *eBACS: ECRYPT Benchmarking of Cryptographic Systems* (2023). Accessed 1 December 2023. URL: https://bench.cr.yp.to. Citations in this document: §1.1, §4.1, §5.1.

[13] Alexandra Boldyreva, Vladimir Kolesnikov (editors), *Public-key cryptography—PKC 2023—26th IACR international conference on practice and theory of public-key cryptography, Atlanta, GA, USA, May 7–10, 2023, proceedings, part I*, 13940, Springer, 2023. ISBN 978-3-031-31367-7. See [17].

[14] Roberta Bragg, *The Encrypting File System* (2009). URL: https://web.archive.org/web/20221007214414/https://learn.microsoft.com/en-us/previous-versions/tn-archive/cc700811(v=technet.10)?redirectedfrom=MSDN#XSLTsection125121120120. Citations in this document: §1.

[15] Mike Cardwell, *Automatically encrypting all incoming email* (2010). URL: https://www.grepular.com/Automatically_Encrypting_all_Incoming_Email. Citations in this document: §1.

[16] Jesse David Dinneen, Ba Xuan Nguyen, *How big are peoples' computer files? File size distributions among user-managed collections*, in ASIST 2021 [**1**] (2021), 425–429. URL: https://arxiv.org/abs/2107.03272. Citations in this document: §6.1, §6.1, §6.1.

[17] Julien Duman, Kathrin Hövelmanns, Eike Kiltz, Vadim Lyubashevsky, Gregor Seiler, Dominique Unruh, *A thorough treatment of highly-efficient NTRU instantiations*, in PKC 2023 [**13**] (2023), 65–94. URL: https://eprint.iacr.org/2021/1352. Citations in this document: §1.1, §1.1, §1.1, §1.1, §1.1.

[18] HTTP Archive, *Report: state of the web* (2023). accessed 1 December 2023. URL: https://httparchive.org/reports/state-of-the-web. Citations in this document: §6.2, §6.2.

[19] Julian Andres Klode, *Encrypted email storage, or DIY ProtonMail* (2019). URL: https://blog.jak-linux.org/2019/06/13/encrypted-email-storage/. Citations in this document: §1.

[20] Jay Ligatti, Xinming Ou, Jonathan Katz, Giovanni Vigna (editors), *CCS '20: 2020 ACM SIGSAC conference on computer and communications security, virtual event, USA, November 9–13, 2020*, ACM, 2020. ISBN 978-1-4503-7089-9. See [25].

[21] Microsoft, *BitLocker FAQ* (2023). URL: https://web.archive.org/web/20230813064046/https://learn.microsoft.com/en-us/windows/security/

operating-system-security/data-protection/bitlocker/faq. Citations in this document: §1.

[22] Ray Perlner, *Re: ROUND 2 OFFICIAL COMMENT: Frodo* (2020). URL: https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/7aenKgDWV2k/m/1SJkfLT9DQAJ. Citations in this document: §6.2.

[23] J. Nathanael Philipp, *Auto encrypt all incoming email with postfix* (2019). URL: https://jnphilipp.org/posts/auto-encrypt-all-incoming-email-with-postfix/. Citations in this document: §1.

[24] Leif Ryge, *Yet another solution to XKCD #949* (2023). URL: https://github.com/leif/dropsite. Citations in this document: §1.

[25] Peter Schwabe, Douglas Stebila, Thom Wiggers, *Post-quantum TLS without handshake signatures*, in CCS 2020 [**20**] (2020), 1461–1480. URL: https://eprint.iacr.org/2020/534. Citations in this document: §6.2.

[26] Statista, *Number of sent and received e-mails per day worldwide from 2017 to 2026* (2023). URL: https://web.archive.org/web/20231127073251/https://www.statista.com/statistics/456500/daily-number-of-e-mails-worldwide/. Citations in this document: §1.2.

[27] Andrew S. Tanenbaum, Jorrit N. Herder, Herbert Bos, *File size distribution on UNIX systems: then and now*, ACM SIGOPS Operating Systems Review **40** (2006), 100–104. URL: https://www.minix3.org/docs/jorrit-herder/osr-jan06.pdf. Citations in this document: §6.1, §6.1.

[28] Jeffrey Scott Vitter (editor), *Proceedings of the thirtieth annual ACM symposium on the theory of computing, Dallas, Texas, USA, May 23–26, 1998*, ACM, 1998. ISBN 0-89791-962-9. See [7].

[29] Garrett Watumull, *Cluster size recommendations for ReFS and NTFS* (2019). URL: https://techcommunity.microsoft.com/t5/storage-at-microsoft/cluster-size-recommendations-for-refs-and-ntfs/ba-p/425960. Citations in this document: §6.1.

[30] Western Digital, *WD Red Pro* (2023). URL: https://web.archive.org/web/20231026192501/https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/product/internal-drives/wd-red-pro-hdd/product-brief-western-digital-wd-red-pro-hdd.pdf. Citations in this document: §2.2, §3.2.