

VANISH

INCREASING DATA PRIVACY WITH SELF-
DESTRUCTING DATA

PRESENTED BY JASON Z. WU

WHY USEFUL

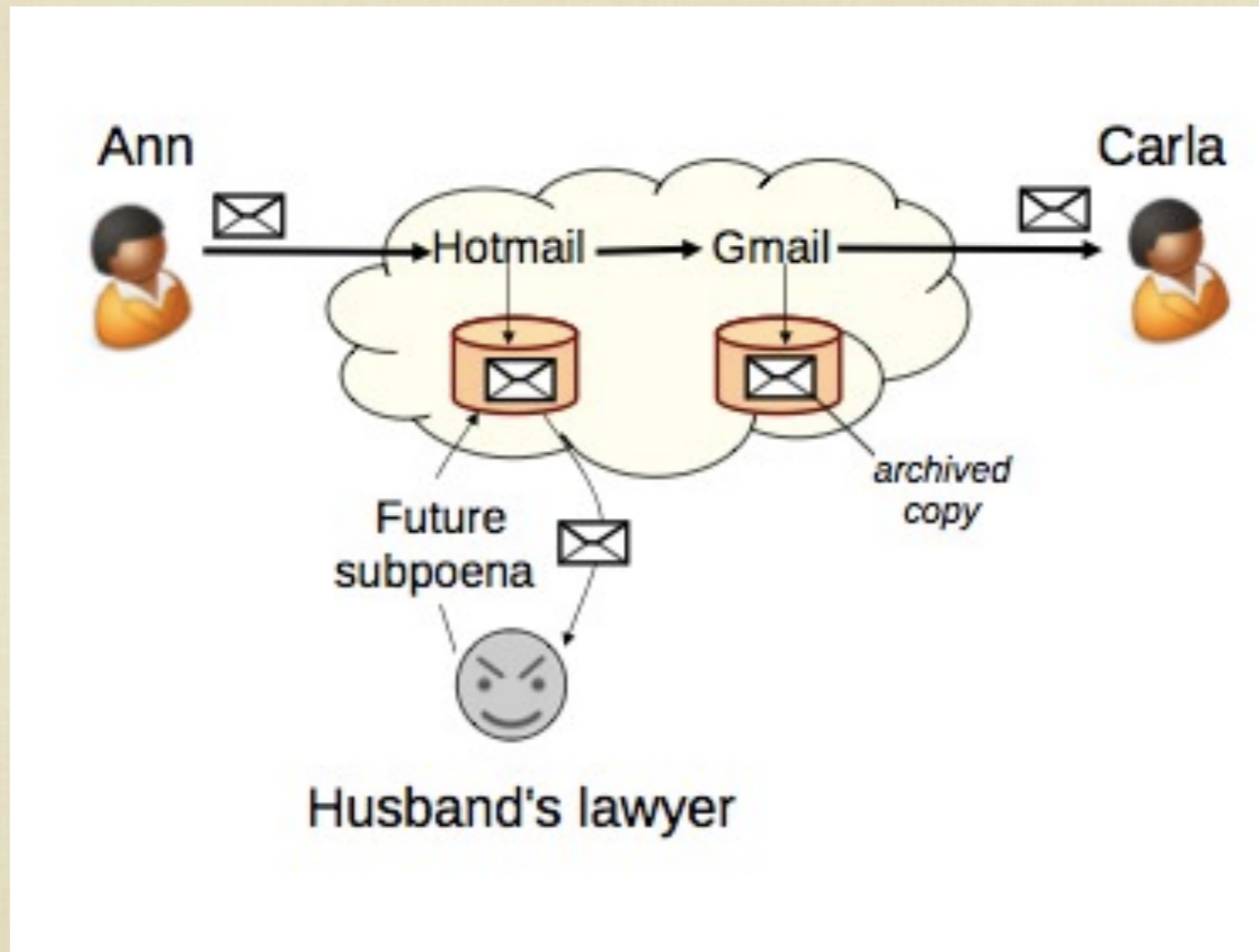
avoid the risk of disclosure to unintended parties

- Emails store on Gmail (cached, stored, may still persist)
- All other private data store on third-party may create a potential risk

from a data sanitation perspective

- many users would benefit from self-destructing trash bins (data will be unavailable to a retroactive examiner)

WHY USEFUL



OBSERVATION & GOAL

create data that self-destructs after a specific time

- automatically (without explicit action by user)
- all copies of data vanish simultaneously (access revoked for everyone, even the legitimate user)
- ❖ Vanish focus on sensitive data (files, private blog posts, online documents), might not be applicable to all data types. User would prefer safety to availability.

Therefore, to regain control over the lifetimes of your data objects both locally and “in the cloud”.

OBSERVATION & GOAL

create a system that can permanently delete data after time-out

- Even if attacker can retroactively obtain a pristine copy before time-out
- Without explicit delete action
- Without modify any of the copies of that data
- Without use of secure hardware
- Without the introduction of any new external services

CANDIDATE APPROACHES (BAD IDEAS)

- Manually Delete
- Public key or symmetric encryption
- Forward-secure encryption
- Deniable encryption
- Ephemeral key exchange for interactive communication systems (eg: OTR)
- Ephemerizer (trusted 3rd parties)

APPROACH

The key insight is to leverage the service provided by decentralized, global-scale P2P infrastructure, in particular, Distributed Hash Tables (DHTs).

(using DHT to implement index-value database on collection of P2P nodes.)

Idea:

▶ to encrypt data with a key, and sprinkle bits of the key across random nodes in the DHT.

WHY DHT?

three unique properties of DHT make them attractive for data destruction goals

- Their huge scale (Robust to influential adversaries)
- Reliable distributed storage (ensure data availability)
- Constantly changing over time (Churn, sprinkle information will naturally disappear)

MODEL & ASSUMPTIONS

Goal: Create a Vanishing Data Object (VDO). A VDO encapsulates user's data and prevents its contents from persisting indefinitely and becoming a source of retroactive information leakage.

Assumptions:

- Time-limited value
- Known time-out
- Internet connectivity
- Dispensability under attack

THREAT MODEL

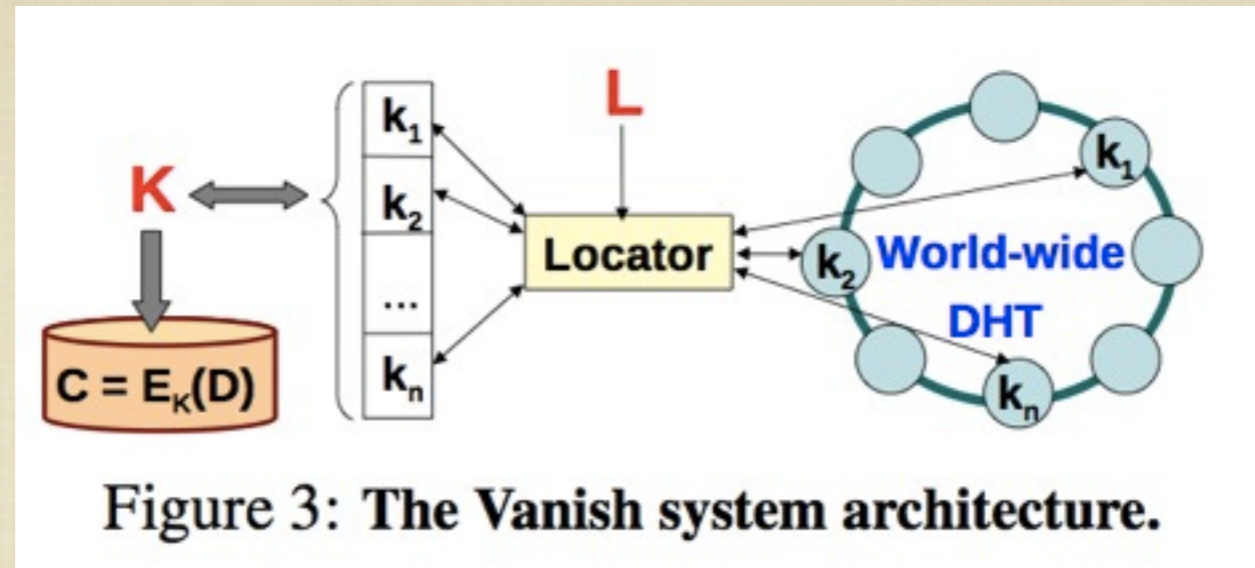
Goal is to ensure the destruction of data after a timeout, despite potential adversaries who might attempt to access that data after its timeout.

we also stress that we explicitly do not seek to preserve goal — accessible prior to a timeout — in the presence of adversaries.

Assumptions:

- Trusted data owners: users with legitimate access to the VDO trust each other, and will not leak the data.
- Retroactive attacks on privacy: attackers do not know which VDOs they wish to access until after the VDOs expire

VANISH ARCHITECTURE



- Vanish takes a data object D (and possibly an explicit timeout T), and encapsulates it into a VDO V .
- to encapsulate the data D , Vanish picks a random data key, K , and encrypts D with K to obtain a ciphertext C . Vanish uses “threshold secret sharing” to split the data key K into N pieces (shares) k_1, \dots, k_n .
- The it picks at random an access key, L . It then uses a secure pseudorandom number generator, keyed by L , to derive N indices into the DHT, i_1, \dots, i_n . Then sprinkles the N shares k_1, \dots, k_n at these nodes throughout the DHT;

VANISH ARCHITECTURE

- Threshold Secret Sharing, Security, and Robustness.

K_1, \dots, K_N will disappear from the DHT over time.

Use a ratio of $< 100\%$, otherwise the loss of a single share would cause the loss of the key.

- Extending the Lifetime of a VDO.

Default timeout offered might be too limiting.

Re-push the pairs periodically? No.

Approach: Do everything again. (retrieve K , re-split, sprinkles)

IMPLEMENTATION

proof of concept Vanish prototype on Vuze DHT and OpenDHT

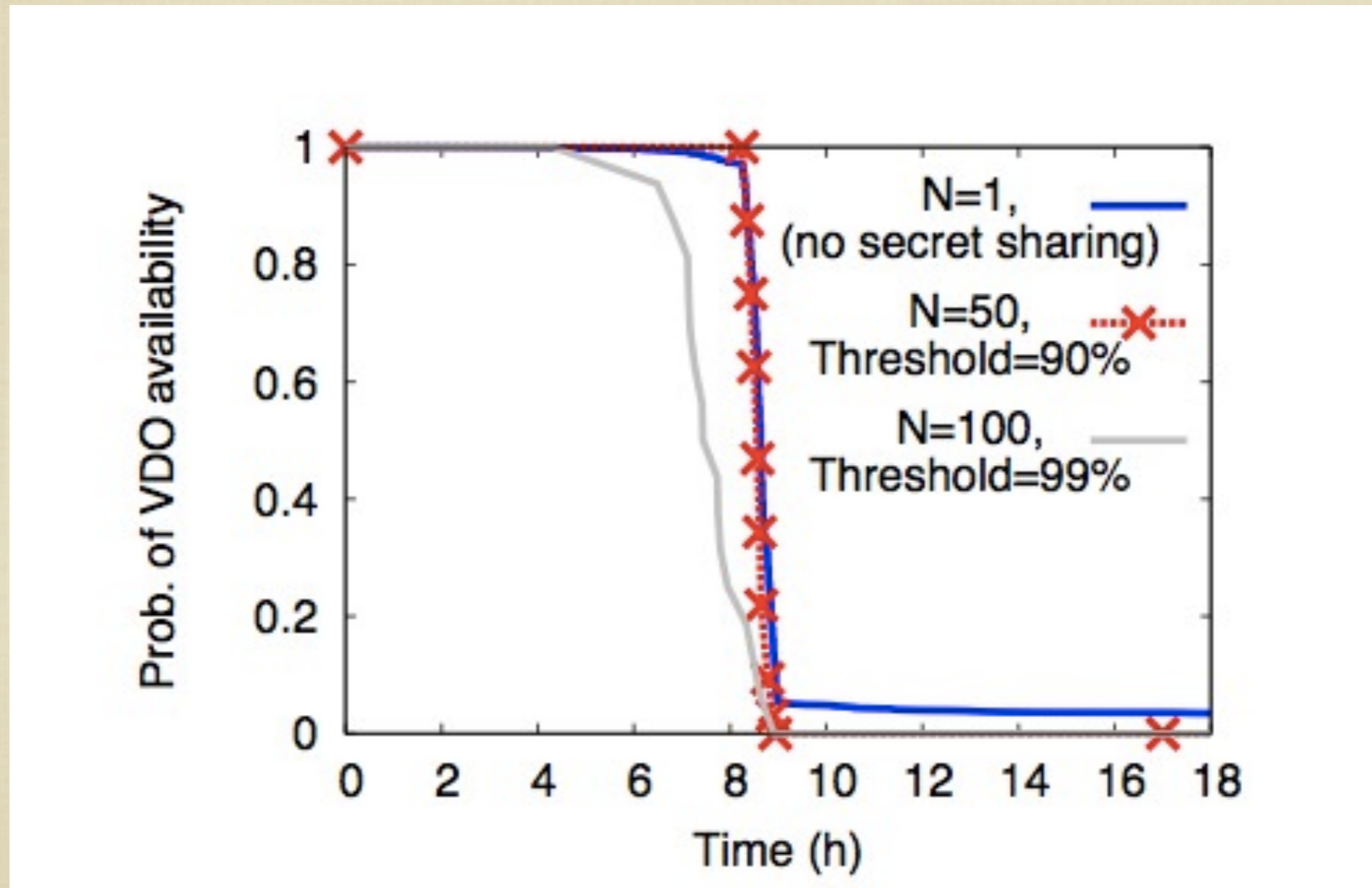
- FireVanish (a Firefox plugin for Gmail and other websites)
Figure 1.
- Vanishing Files (a self destructing file manager)

IMPLEMENTATION

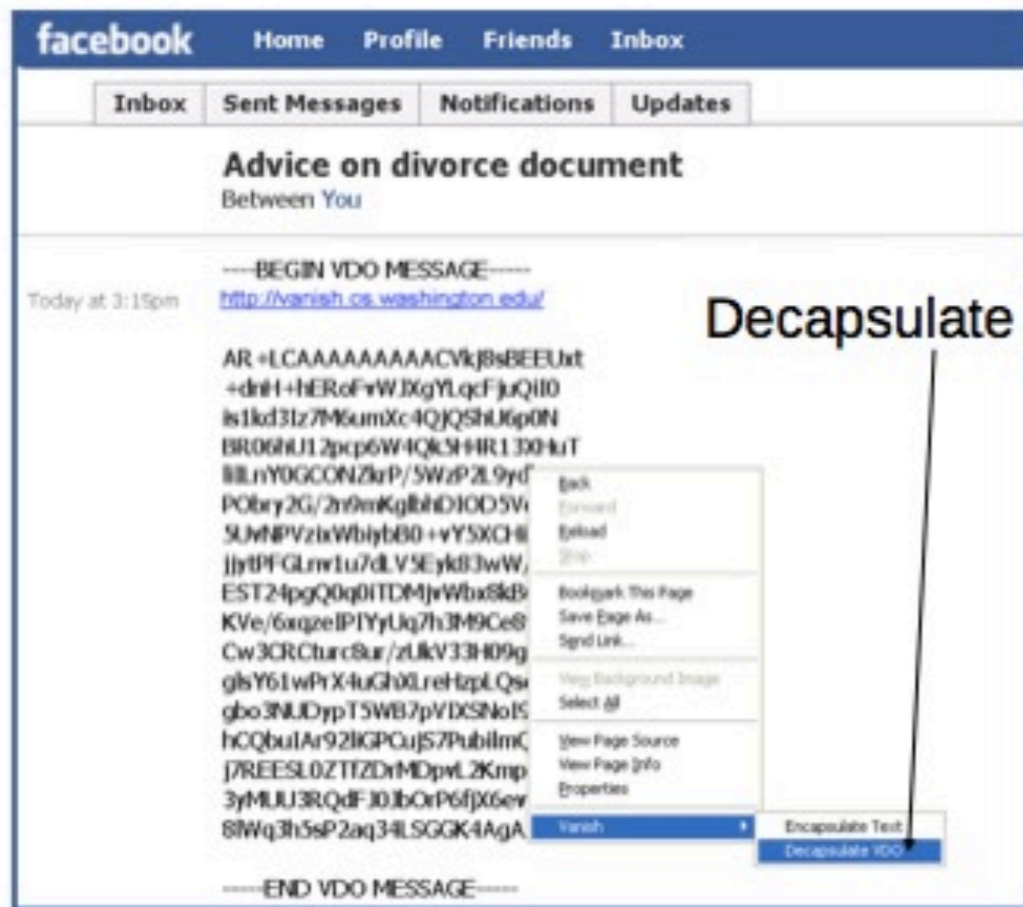
Firefox plugin for web-based email (e.g. Gmail)

- Compose email, encrypt with random key
- Split key
- Throw away the key
- Publish the shares to random DHT nodes
- Send encrypted email and key share locator L
- Recipient collects secret shares to decrypt

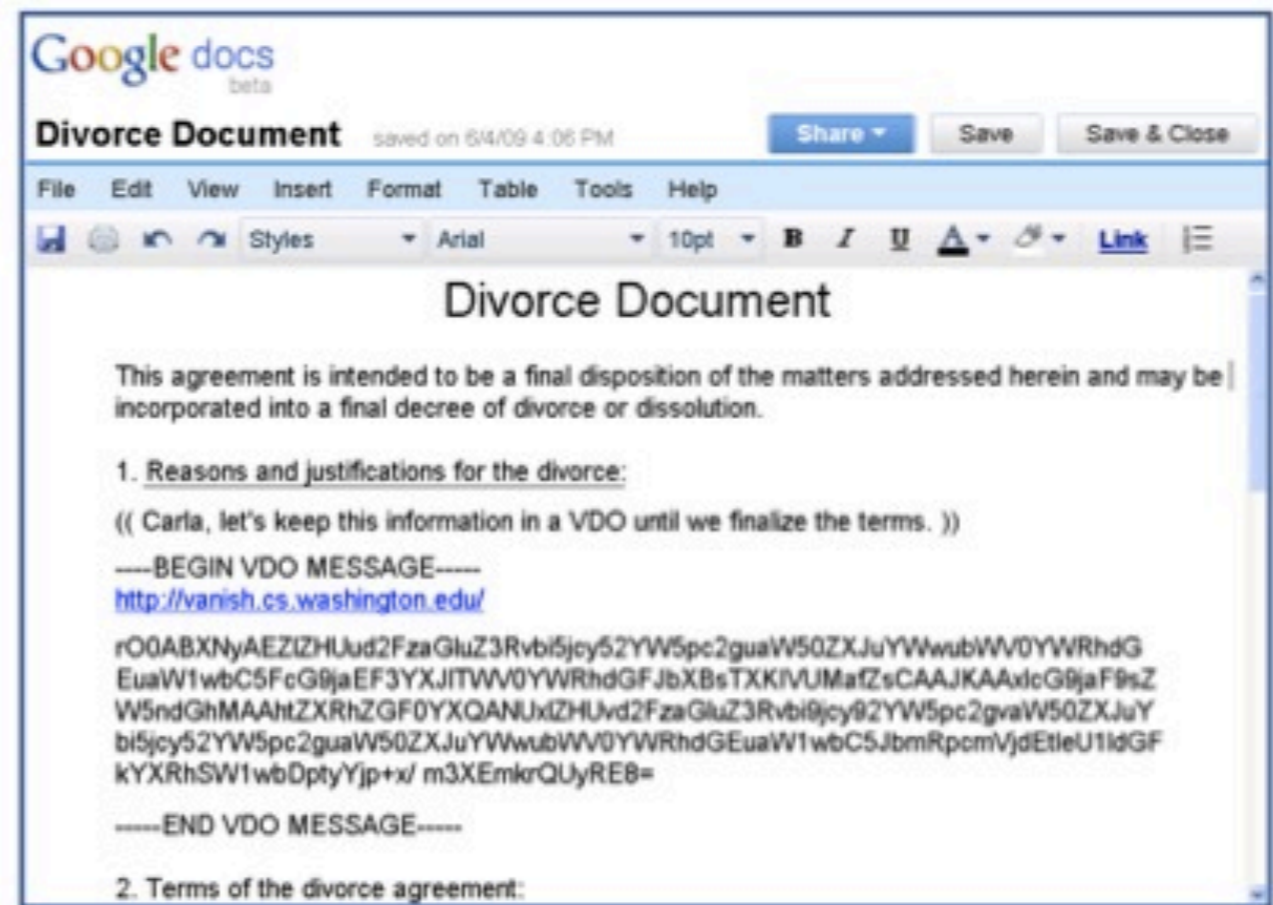
AVAILABILITY OF KEY SHARES



INTERFACE



(a) Vanishing Facebook messages.

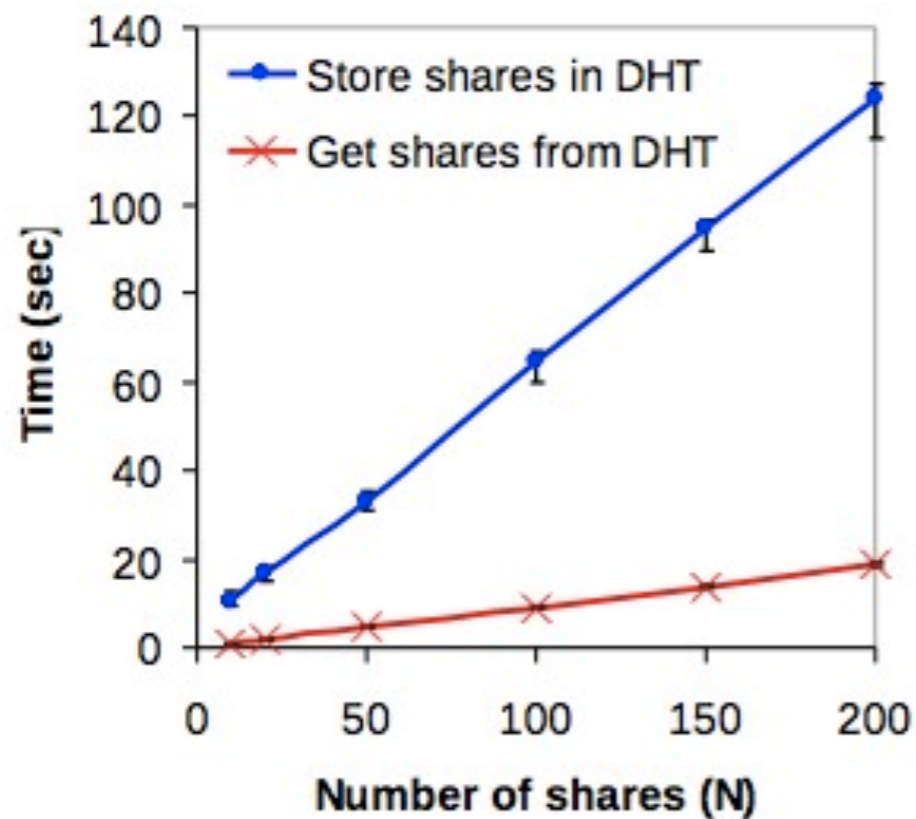


(b) Google Doc with vanishing parts.

INTERFACE



PERFORMANCE



(a) Scalability of DHT operations.

N	Time (seconds)		
	Encapsulate VDO		Decapsulate VDO
	Without prepush	With prepush	
10	10.5	0.082	0.9
20	16.9	0.082	2.0
50	32.8	0.082	4.7
100	64.5	0.082	9.2
150	94.7	0.082	14.0
200	124.3	0.082	19.0

(b) VDO operation execution times.

POTENTIAL ATTACKS

- Decode before shares expire
- Sniff user's Internet connection
- Become part of the DHT – an adversary would need to join the 1M-node Vuze DHT with approximately 80,000–90,000 malicious nodes to mount a store-based attack and capture a reasonable percentage of the VDOs (e.g., 25%)

THANKS!