Parallelized Common Factor attack on RSA

Aneek Roy

BCSE Final Year

Roll - 001410501072

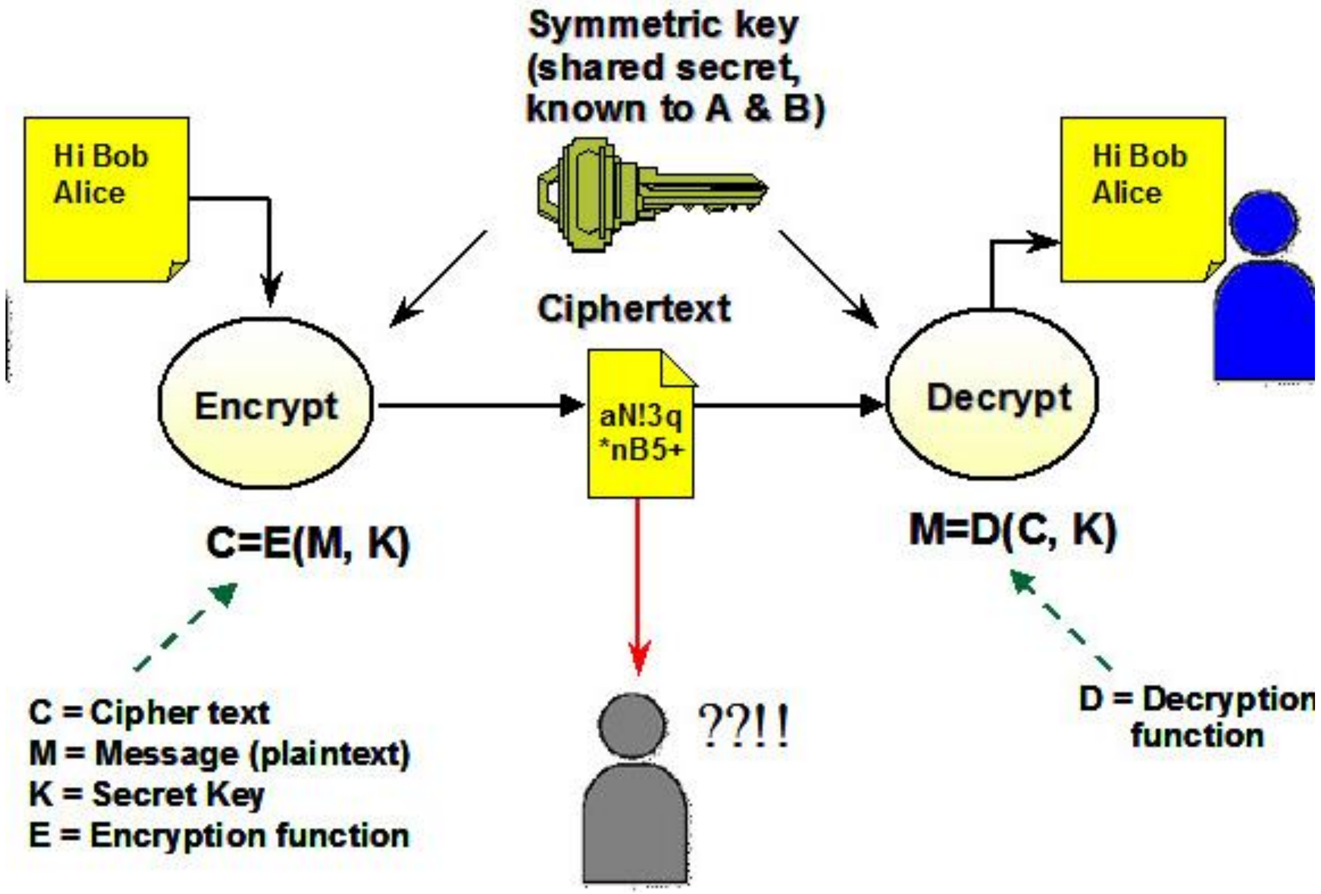
Types of Cryptosystems :

- Symmetric Key Cryptography

- Asymmetric Key Cryptography

Symmetric Key Cryptography

- Example of implementations :
- DES[2], 3-DES[3] , AES[4], IDEA, and BLOWFISH



Problems of Symmetric key Cryptography :

- **Key establishment** − Before any communication, both the sender and the receiver need to agree on a secret symmetric key. It requires a secure key establishment mechanism in place, usually provided by asymmetric key cryptography.

- **Trust Issue** − Since the sender and the receiver use the same symmetric key, there is an implicit requirement that the sender and the receiver ‘trust’ each other. If an attacker gets hold of the shared private key, all secrecy in the message passing is lost.

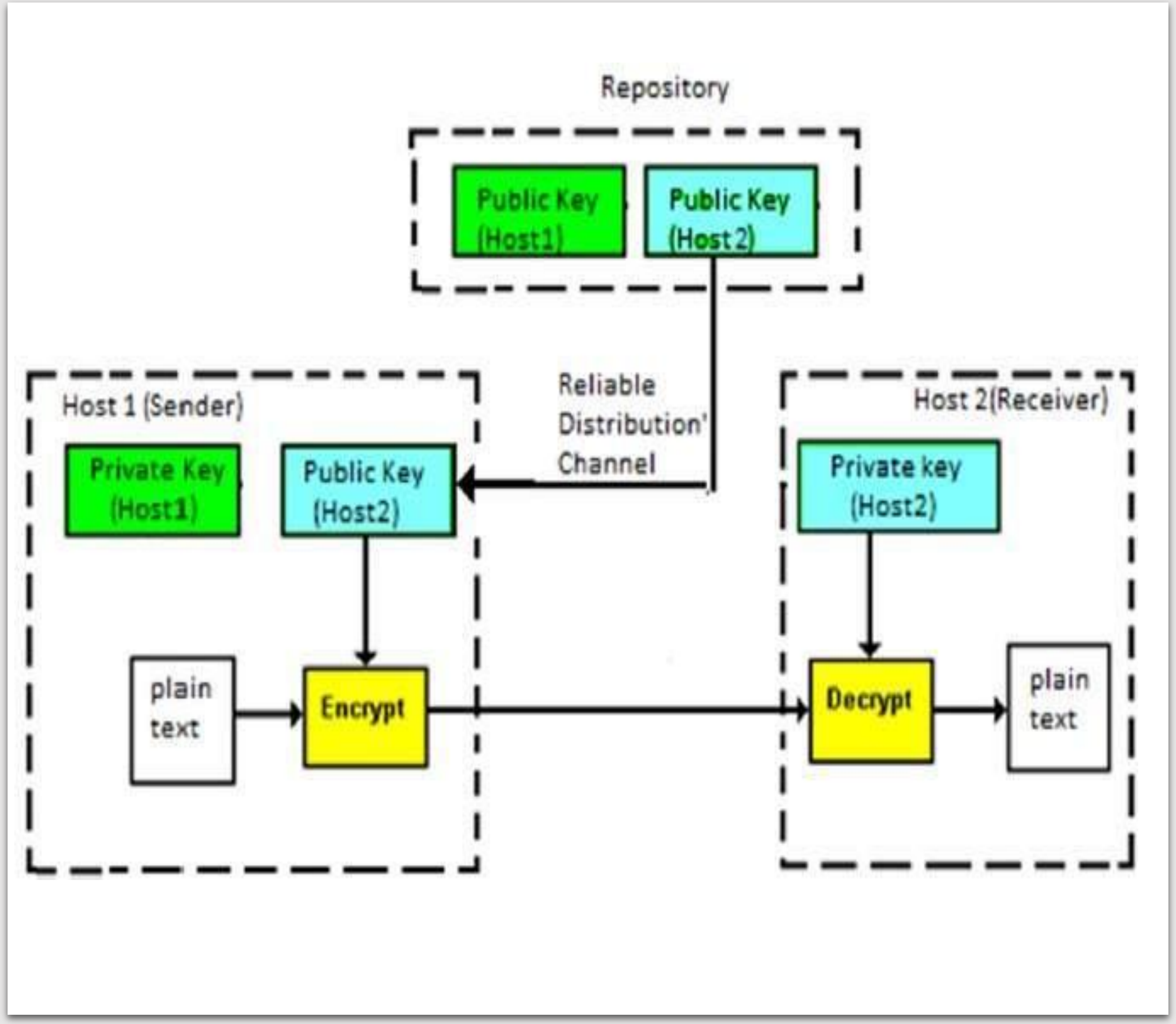
Asymmetric key Cryptography :

- First proposed as a theoretical model by Diffie and Hellman[1].
- Some Properties include :
- Decryption and Encryption are inverse functions to each other that is:

D(E(M)) = M as well as E((D(M))=M

- D and E functions are easy to compute
- By publicly revealing D, the user does not reveal an easy way to compute E.
- Some implementations include :

RSA [5] , ElGamal [6] and Diffie-Hellman Key Exchange [1] program



RSA [5] Encryption System

- RSA was the first practical model suggested by Rivest,Shamir and Adelman implementing public key encryption system.
- A user Alice has a set of public key (n,e) and a private key (n,d).
- Here n is the product of 2 "random" large primes : n = p.q
- The integer d is chosen such that gcd(d,(p-1).(q-1)) = 1
- Thus we get e which is a multiplicative inverse of d modulo(p-1)(q-1)

  e.d = 1 (mod (p-1)(q-1))
- If any user wants to send any data to Alice, he encrypts the Message using public key of Alice.
- Thus the Ciphertext C is generated by : C = E(M) = M^e (mod n)
- The Message can be retrieved by Alice through the Decryption function implemented using her private key : M = D(C) = C^d (mod n)

Problems in implementation of RSA algorithm

- The security of RSA lies in factoring of large integers composed of large primes being a hard problem. If n can be factored into p,q then private key d can be easily recovered.

- In theory, if two primes p and q are chosen uniformly at random from all possible 512-bit primes, then the chance of getting the same prime twice is approximately 2^{-256} (birthday collision probability).

- This idea is challenged by Heninger et al. [10] & Lenstra et al. [9]

Idea of Common Factor Attack as Proposed by Heninger et al.[10]

- $N_1 = p.q_1$
- $N_2 = p.q_2$
- For 1024-bit RSA, it is expected that the primes p and q are chosen independently and uniformly at random from all possible 512-bit primes, satisfying p != q
- Heninger et al. [10] traced the cause of this vulnerability to sloppy implementations of RSA in embedded systems, especially in routers, firewalls, and other network devices. In case of random prime generation, RSA implementations tend to use pseudo-random number generators (PRNGs).
- However, as the smaller network devices try to generate the primes at boot, quite often they lack a full-entropy source for extracting the random seeds,and hence the primes they generate eventually have a much higher probability of collision.

Idea of Common Factor Attack as Proposed by Heninger et al.[10] using Bernstein's[8] factoring algorithm.

- Compute the Product of all RSA moduli P=πNi, using a binary tree of partial products.
- Compute Zi = P mod Ni^2 using remainder tree.
- Compute gcd(Zi/Ni,Ni) to the list of all vulnerable RSA modulis.

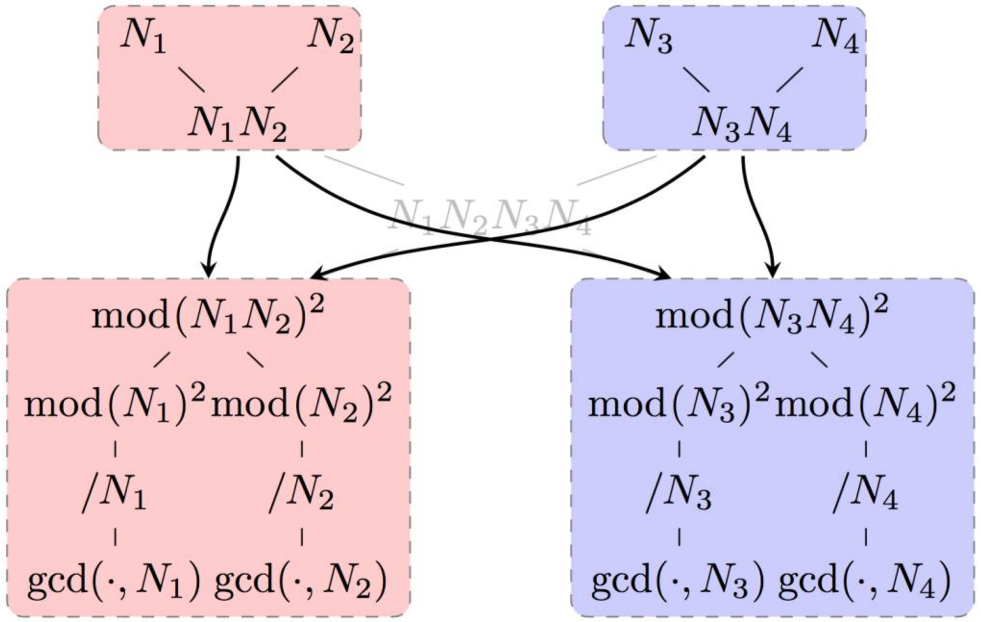
$$\begin{tikzcd}
& N_1 \arrow[dr] & & N_2 \arrow[dl] & & N_3 \arrow[dr] & & N_4 \arrow[dl] & \\
& & \times \arrow[drr] & & & & \times \arrow[dll] & & \text{product tree} \\
& & & & N_1N_2N_3N_4 \arrow[dll] \arrow[drr] & & & & \\
& & \bmod N_1^2N_2^2 \arrow[dl] \arrow[dr] & & & & \bmod N_3^2N_4^2 \arrow[dl] \arrow[dr] & & \text{remainder tree} \\
& \bmod N_1^2 \arrow[d] & & \bmod N_2^2 \arrow[d] & & \bmod N_3^2 \arrow[d] & & \bmod N_4^2 \arrow[d] & \\
& /N_1 \arrow[d] & & /N_2 \arrow[d] & & /N_3 \arrow[d] & & /N_4 \arrow[d] & \\
& \gcd(\cdot, N_1) & & \gcd(\cdot, N_2) & & \gcd(\cdot, N_3) & & \gcd(\cdot, N_4) &
\end{tikzcd}$$

Results :

- Used batchwise GCD algorithm to find out vulnerable moduli.Found that 0.75% of TLS certificates shared RSA primes, and conjectured that another 1.70% were susceptible to compromise.
- But required almost 32 GB of memory and around 60 to 70 GB of storage for scratch calculations.Thus there is a HUGE Computational Bottleneck.

Method Proposed by Hastings et al. [7]

- Proposed a partially parallel implementation of batch-wise GCD
- Dataset is partitioned in subsets and the product tree for computed such that Pi = π Ni is constructed individually for each subset.
- Provides full parallelization in the first phase of the algorithm, the remainder tree is still constructed considering all subset products produced by the product trees.
- However, enormous Memory and Computational resource was used. They used a quad 6-core 3.40 Ghz Intel Xeon E7-8893 Processors with 3 TB RAM and over 500 GB of memory to implement their parallelised approach.



Is there a better way to implement the Batch-wise GCD algorithm in a resource constrained environment ?

- Divide dataset randomly into p parts, where p ~ |D|/|m| , D is the size of the large dataset and m is the size of the dataset which can fit into the resource constrained unit. Thus |D|>>|m|.
- We apply the batch-wise GCD algorithm over each partitions separately.
- Obviously there will be instances where the gcd(Ni,Nj) > 1 and Ni, Nj are in separate partitions.
- To overcome this, we use multiple randomly split partitions of the dataset, run batchwise GCD and aggregate the results.
- The relationship between number of partitions p , the level of accuracy desired epsilon(€) and the number of iterations k can be formalised through theorem 1 stated later in the slides.

```
Input   : Set of moduli $D$, constraint $m$, accuracy $\epsilon$
Output: $V$ — set of vulnerable moduli in $D$
1 $p \leftarrow$ ceiling($|D|/m$) ;
2 $k \leftarrow$ chooseIteration($m$, $p$, $\epsilon$) ;
3 for $i \leftarrow 1$ to $k$ do
4     $\{d_1, d_2, \ldots, d_p\} \leftarrow$ randomPartition($D$, $p$) ;
5     $\{v_1, v_2, \ldots, v_p\} \leftarrow$ batchGCD($\{d_1, d_2, \ldots, d_p\}$) ;
6     $V_i \leftarrow$ setUnion($\{v_1, v_2, \ldots, v_p\}$) ;
7 end
8 $V \leftarrow$ setUnion($\{V_1, V_2, \ldots, V_k\}$) ;
```

Algorithm 1: Parallelized Common Factor Attack

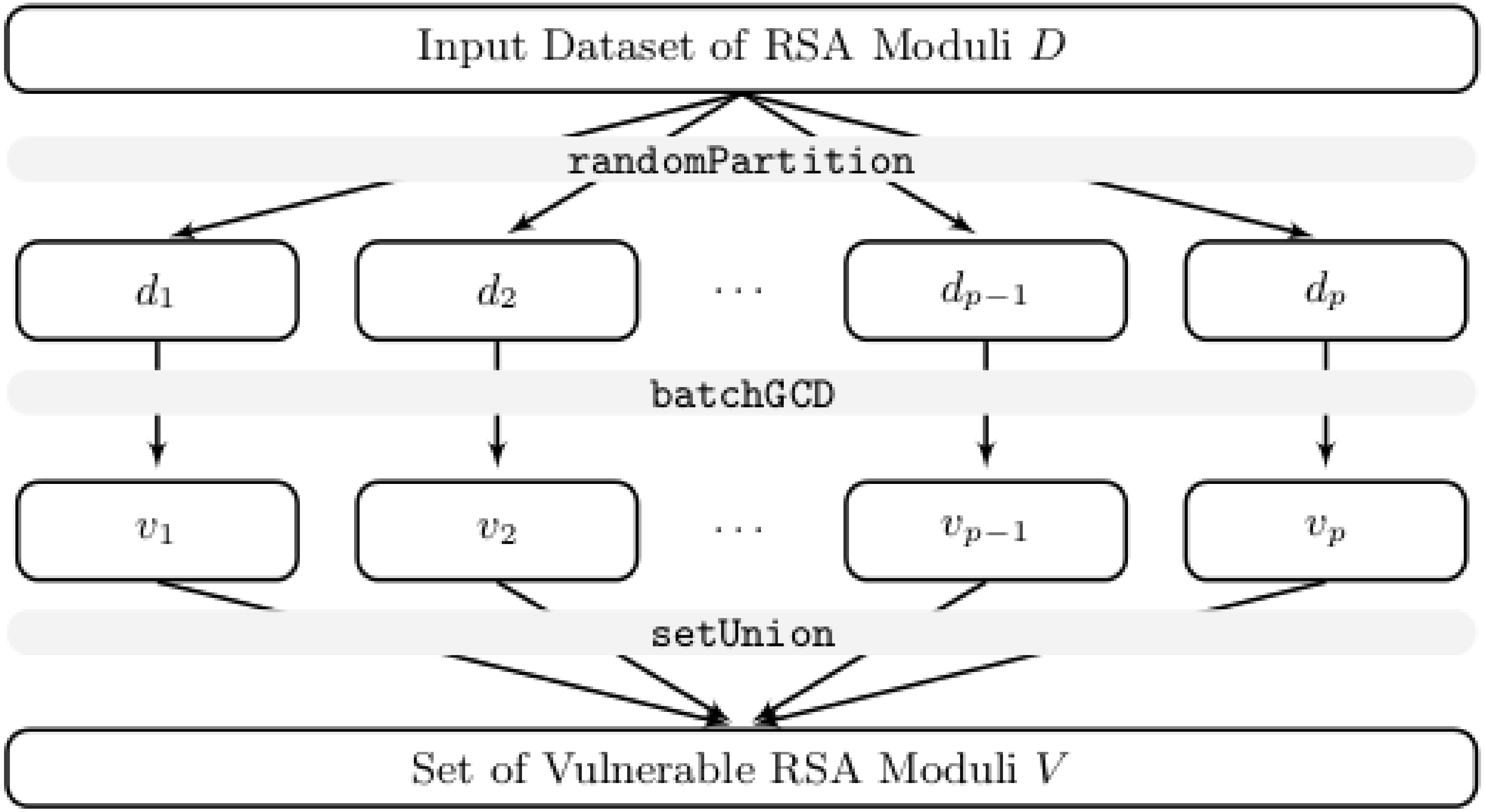


Fig. 2: One complete iteration of the proposed Parallelized Algorithm.

Theorem

Suppose there exist X vulnerable RSA moduli in input dataset D. Then Algorithm 1 recovers an expected number of ϵX vulnerable moduli if we set

$$k \approx \frac{\log(1-\epsilon)}{\log m + \log(p-1) - \log(mp-1)},$$

where ϵ is the user-defined accuracy parameter, m is the user-defined constraint of the individual computing nodes, and $p \sim |D|/m$ is the number of partitions.

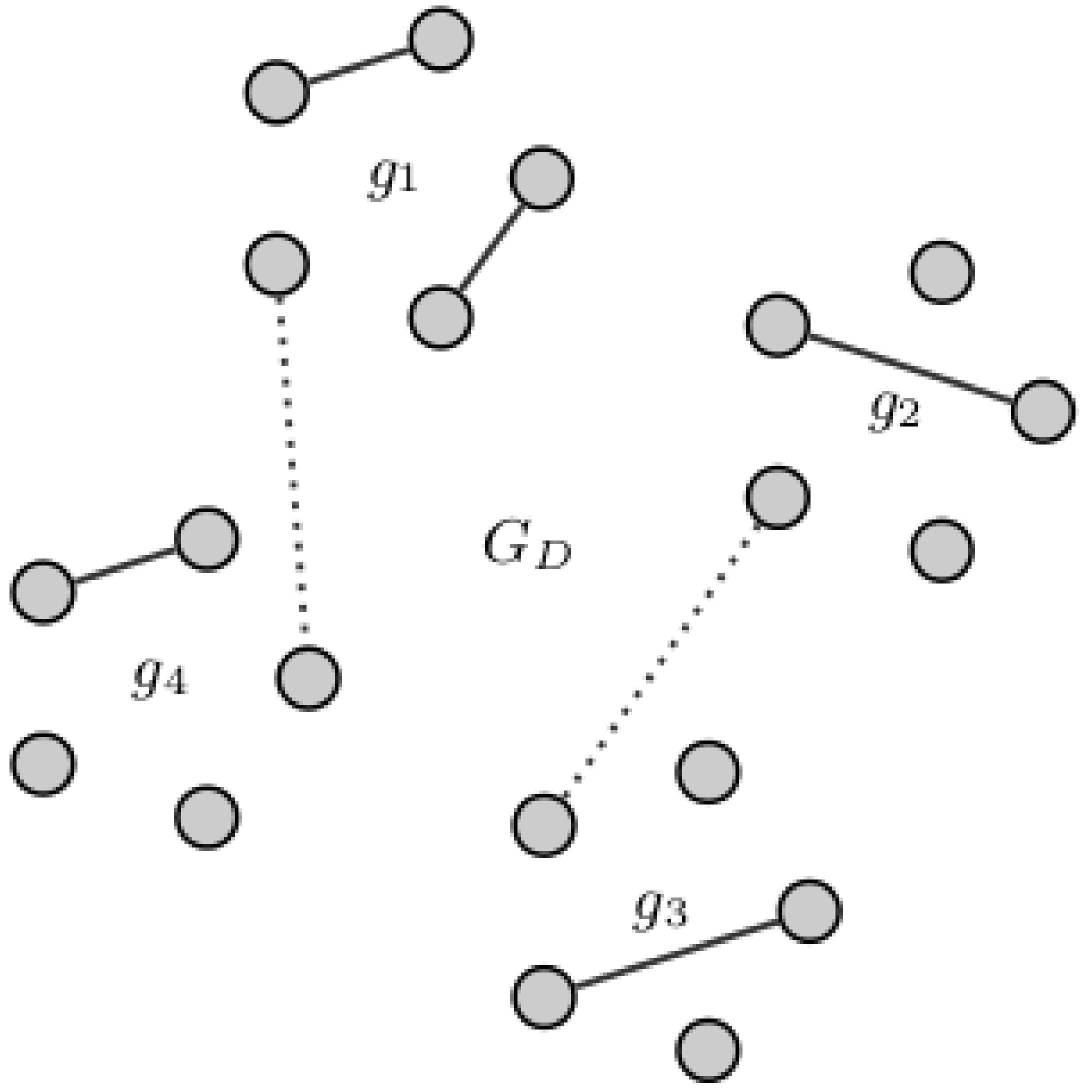


Fig. 3: Illustrative partition of graph G_D into subgraphs $\{g_1, g_2, \ldots, g_p\}$.

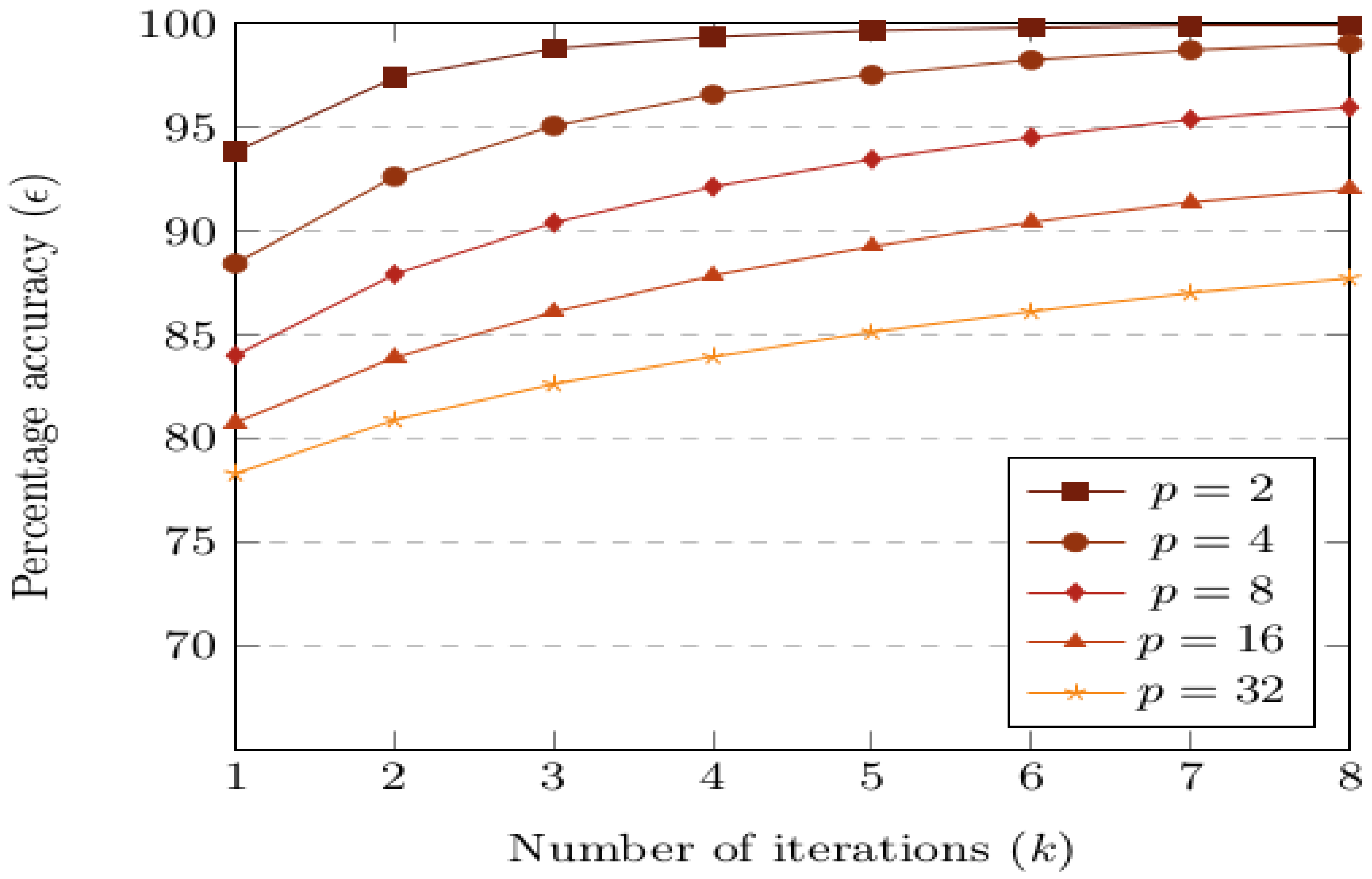


Fig. 4: Relationship between ϵ, p and k from experimental data.

References

1. Diffie, Whitfield, and Martin Hellman. "New directions in cryptography." *IEEE transactions on Information Theory* 22.6 (1976): 644-654.
2. Diffie, Whitfield, and Martin E. Hellman. "Special feature exhaustive cryptanalysis of the NBS data encryption standard." *Computer* 10.6 (1977): 74-84.
3. William C. Barker and Elaine B. Barker. 2012. *SP 800-67 Rev. 1. Recommendation for the Triple Data Encryption Algorithm (Tdea) Block Cipher*. Technical Report. NIST, Gaithersburg, MD, United States.
4. Rijmen, Vincent, and Joan Daemen. "Advanced encryption standard." *Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology* (2001): 19-22.
5. Rivest, Ronald L., Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems." *Communications of the ACM* 21.2 (1978): 120-126.

References

6. ElGamal, Taher. "A public key cryptosystem and a signature scheme based on discrete logarithms." *IEEE transactions on information theory* 31.4 (1985): 469-472.

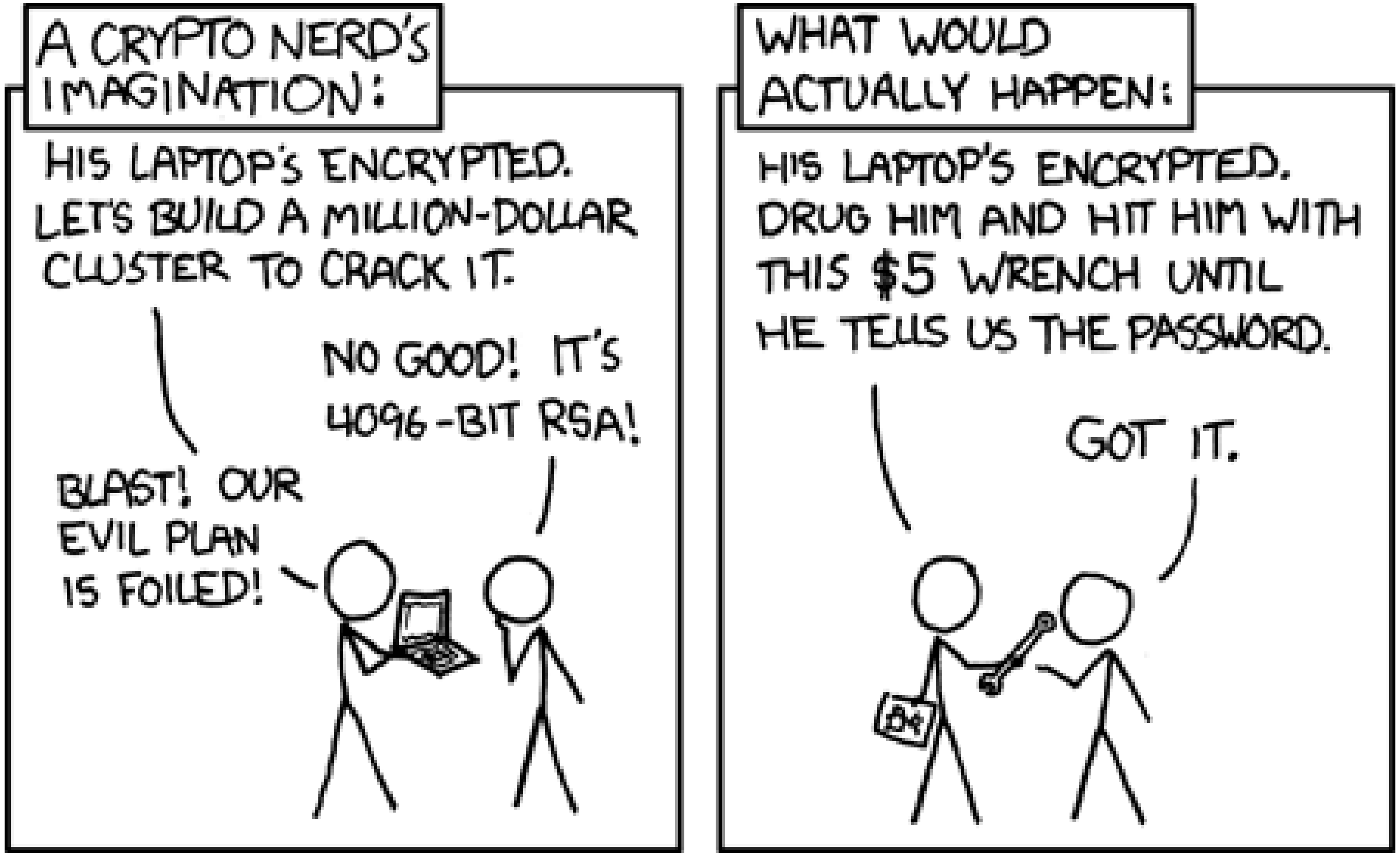
7. Hastings, Marcella, Joshua Fried, and Nadia Heninger. "Weak keys remain widespread in network devices." *Proceedings of the 2016 ACM on Internet Measurement Conference*. ACM, 2016.

8. Bernstein, Daniel J. "How to find smooth parts of integers." *URL: http://cr. yp. to/papers. html# smoothparts. ID 201a045d5bb24f43f0bd0d97fcf5355a. Citations in this document* 20 (2004).

9. Lenstra, Arjen, et al. *Ron was wrong, Whit is right*. No. EPFL-REPORT-174943. IACR, 2012.

10. Heninger, Nadia, et al. "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices." *USENIX Security Symposium*. Vol. 8. 2012.

Thank you !


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.