The Hidden Number Problem with Small Unknown Multipliers: Cryptanalyzing MEGA in Six Queries and Other Applications

Nadia Heninger and Keegan Ryan(✉)

University of California, San Diego, USA
nadiah@cs.ucsd.edu, kryan@eng.ucsd.edu

Abstract. In recent work, Backendal, Haller, and Paterson identified several exploitable vulnerabilities in the cloud storage provider MEGA. They demonstrated an RSA key recovery attack in which a malicious server could recover a client's private RSA key after 512 client login attempts. We show how to exploit additional information revealed by MEGA's protocol vulnerabilities to give an attack that requires only six client logins to recover the secret key.

Our optimized attack combines several cryptanalytic techniques. In particular, we formulate and give a solution to a variant of the hidden number problem with small unknown multipliers, which may be of independent interest. We show that our lattice construction for this problem can be used to give improved results for the implicit factorization problem of May and Ritzenhofen.

1 Introduction

MEGA is an encrypted cloud storage provider whose protocols are designed to protect a client's data and secret key against a malicious server or malicious entity in the backend infrastructure. In a recent paper [2], Backendal, Haller, and Paterson detail multiple exploitable flaws in MEGA's protocols including a full key recovery attack [2, Section III].

In this attack, a malicious MEGA server uses a victim client as a decryption oracle to learn information about mauled encryptions of the user's private RSA key. In Backendal, Haller, and Paterson's original work, the attacker learns one bit of information about the key per query, and thus needs at least 512 client login attempts to recover enough information to efficiently recover the rest of the secret RSA key.

We observe that the attacker can learn up to 43 bytes of information per query, and give an algorithm to efficiently exploit this information for an improved attack that only requires six login attempts from a victim client. This brings the attack into a much more realistic range of failed login attempts that would be tolerated by a human user.

A. Boldyreva and V. Kolesnikov (Eds.): PKC 2023, LNCS 13940, pp. 147–176, 2023.
https://doi.org/10.1007/978-3-031-31368-4_6

Our work does not exploit any new vulnerabilities in MEGA's protocol; instead, we show that the risk to unpatched clients was significantly underestimated in [2]. MEGA patched the original issue by adding additional payload validation and emphasized in their blog post about the vulnerabilities [24] that only clients who have logged in more than 512 times are vulnerable. Their patch is effective at preventing our attack as well.

1.1 Technical Overview

Our full attack exploits the interplay between the symmetric and asymmetric cryptographic operations in MEGA's design and has several stages.

We wish to draw the reader's attention to one new subproblem that our new attack needed to solve, which we had not seen articulated before in the literature. We call this problem the Hidden Number Problem with Small Unknown Multipliers (HNP-SUM).

Definition 1 (HNP with Small Unknown Multipliers). *Given integer inputs N, a_i, T, and E such that for $1 \leq i \leq n$ there exist integers x, t_i, e_i satisfying*

$$\begin{aligned} a_i &\equiv t_i x + e_i \pmod{N} \\ |t_i| &\leq T \\ |e_i| &\leq E, \end{aligned}$$

the goal of the adversary is to recover the vector of t_i values up to sign and common divisor.

The reason we only require recovery of the collection of $t_1, \ldots, t_n$ up to sign and common divisor g is because if $t_1, \ldots, t_n, x$ satisfy the HNP-SUM equations, then so do $-t_1, \ldots, -t_n, -x$ and $t_1/g, \ldots, t_n/g, gx$.

This problem can be viewed as a variant of the hidden number problem in which the multipliers t_i are unknown, but known to be small, or a variant of the approximate GCD problem [15] with an additional modular reduction step with an unrelated modulus N.

We develop an efficient lattice-based approach that solves HNP-SUM heuristically. We apply our approach to the cryptanalysis of MEGA, enabling the attack to succeed in as few as six login attempts with high probability. Our definition of HNP-SUM also leads to a new approach for solving the implicit factoring problem, first introduced by May and Ritzenhofen in PKC 2009 [21].

Theorem 1. *Let N, a_i, T, E, and n define an instance of HNP-SUM as in Definition 1 with t_i generated uniformly at random. There exists a heuristic polynomial time algorithm that solves HNP-SUM when*

$$T^{(n+1)/(n-1)} E \lesssim N.$$

Although the exponential approximation factor of polynomial time lattice reduction algorithms does influence these bounds, the effect is minimal for small to medium n. A bound with additional terms is explored in Lemma 4.

Our lattice construction has dimension $n+1$ and entries of bit length $\log_2 N$. It is based on the observation that $t_2 a_1 - t_1 a_2 \equiv t_2 e_1 - t_1 e_2 \pmod{N}$ is a small integer linear combination modulo N that can be found by lattice reduction. There is a natural way to extend the lattice construction for $n > 2$, but the analysis becomes substantially more involved. To successfully recover the t_i, we must first use lattice reduction to find a basis for a dense sublattice of rank $n-1$, and we then use the Hermite Normal Form to calculate the t_i from the sublattice. We analyze the sublattice structure heuristically to derive the bound in Theorem 1.

We give our detailed analysis in Sect. 3. We first analyze the $n = 2$ case, and we then extend our approach for $n > 2$.

1.2 Applying HNP-SUM to MEGA Cryptanalysis

In the MEGA attack context of [2], the server possesses an encrypted copy of the user's RSA private key, which is encrypted using AES-ECB with a key derived from the user's password. Encrypting the RSA private key held by the server is meant to stop a malicious server from decrypting the user's files while allowing a user to log in on a new client and decrypt their own files using only a password.

During the login process, the server sends the user the symmetrically encrypted RSA private key and a challenge ciphertext encrypted to the user's RSA public key. The user decrypts the RSA private key using their password, decrypts the challenge RSA ciphertext with the RSA private key they just decrypted, and responds to the server with the most significant bytes of the RSA plaintext. If these bytes match the server's RSA plaintext, this is intended to confirm that the user knows the RSA private key and therefore knows the password.

In the attack of Backendal, Haller, and Paterson, a malicious server uses a victim client as a decryption oracle to recover the symmetrically wrapped private RSA key. During the login procedure, the malicious server can send the user a mauled version of the user's encrypted private key. Because the private key is encrypted with ECB mode, it is possible for the attacker to selectively modify specific fields in the private key, and the victim client uses this maliciously modified private key to decrypt the server's RSA ciphertext c. The encrypted private key is a custom data structure encoding the prime factors p and q of N, the decryption exponent d, and the RSA-CRT coefficient $u \equiv q^{-1} \pmod{p}$.

The value that the client sends back to the server is

$$\mathrm{MSB}((u(m_p - m_q)q + m_q) \bmod N)$$

where m_p and m_q are $(c^d \bmod p)$ and $(c^d \bmod q)$ respectively.

In our attack, the attacker swaps ciphertext blocks into the location encoding u, and observes the client's decrypted values to learn information about u. In a toy simplification of our attack, consider the case where block swaps result in u being a single 128-bit block of AES plaintext. This gives an instance of HNP-SUM; u is the small unknown multiplier, $(m_p - m_q)q$ is the hidden number, and the most significant bytes after shifting by m_q give the approximation.

Solving HNP-SUM reveals the value of the unknown multiplier u and thus the corresponding block of AES-encrypted plaintext, providing the attacker with an AES decryption oracle that can be used to decrypt arbitrary blocks of the symmetrically encrypted RSA private key, eventually revealing the entire key.

In the actual attack, the attacker is significantly more constrained, and we detail the multiple steps that make it possible to perform the block swapping attack and obtain HNP-SUM samples that are sufficiently bounded and recoverable. We detail a fast attack that recovers the victim's RSA private key with 16 login attempts in a few seconds with success rate 93.9% and a small attack that recovers the victim's key with 6 login attempts in 4.5 h with success rate 97.7%.

Disclosure. The details of the attack of Backendal, Haller, and Paterson and associated patches were made public June 21, 2022. We notified MEGA of the improved cryptanalysis on July 13, 2022, which they acknowledged immediately after in an update to their blog post. In response to our improved cryptanalysis, they have updated their guidance to acknowledge that the potential exposure applies to "the vast majority of users" and underscore the importance of installing the patch.

1.3 Applying HNP-SUM to Implicit Factoring

In the implicit factoring problem [21], the adversary is given k (unbalanced) RSA moduli of the form $N_i = p_i q_i$, where the p_i share some bits in common. The bit length of q_i is α, and t is the number of bits shared by the p_i. Depending on the variant of the implicit factoring problem, these t bits may be least significant bits, most significant bits, a mix of both, or a block of t contiguous bits in the middle of p_i [27].

All four cases can be reduced to instances of HNP-SUM. When the t middle bits at offset l are shared, we have $p_i = 2^{l+t}\tilde{p}_i' + 2^l p_{mid} + \tilde{p}_i$, giving

$$N_i \equiv q_i(2^l p_{mid}) + (q_i \tilde{p}_i) \pmod{2^{l+t}}.$$

This is an instance of HNP-SUM with small unknown multiplier q_i, hidden number $2^l p_{mid}$, and approximation $N_i \bmod 2^{l+t}$. Solving HNP-SUM reveals q_i, which reveals the factorization of the N_i.

Our reduction to HNP-SUM gives a new lattice-based approach to solving the implicit factoring problem. We find that the case of shared middle bits can be heuristically solved when $t \geq 2\frac{k}{k-1}\alpha$ and the other three cases can be solved when $t \geq \frac{k}{k-1}\alpha$. For the shared middle bit case, our construction is significantly more efficient than the existing literature, since it requires a lattice dimension of $k+1$, where prior work used a lattice of dimension $\Theta(k^2)$. Our bounds match the heuristic bounds of existing non-Coppersmith lattice-based solution methods [9, 21], but do not improve the bounds of the Coppersmith-based approaches [20, 25, 27–29]. In addition, our lattice approach is the first non-Coppersmith approach to our knowledge that solves the mixed least significant and most significant bits case.

2 Background

2.1 Lattices

Our algorithms make use of integer lattices in multiple ways. Given a collection of vectors $B \subset \mathbb{Z}^m$, the lattice $\mathcal{L}(B) = \{\sum_{\vec{b}_i \in B} c_i \vec{b}_i \mid c_i \in \mathbb{Z}\}$ is the set of integer linear combinations of vectors in B. The dimension of the lattice is m. The rank d of the lattice is the rank of the matrix B. In this work, we represent basis vectors as the rows of the matrix $B \in \mathbb{Z}^{d \times m}$. A lattice does not have a unique basis, but the lattice determinant, calculated as $\det(\mathcal{L}(B)) = \sqrt{\det(BB^T)}$ is an invariant of the lattice.

The Hermite Normal Form (HNF) of a lattice $\mathcal{L}$ is a unique representation for a basis for $\mathcal{L}$. The HNF is upper triangular, the elements along the diagonal, known as the pivots, are positive, and elements above pivots are positive and bounded above by the pivot. Given any basis B, it is possible to compute $\mathrm{HNF}(\mathcal{L}(B))$ in polynomial time [18].

The successive minima $\lambda_i(\mathcal{L}(B))$ of a lattice are defined as the lengths of the i shortest linearly independent vectors in $\mathcal{L}(B)$. The Gaussian Heuristic [10] predicts that, for a random lattice, the successive minima approximate

$$\lambda_i(\mathcal{L}(B)) \approx \sqrt{\frac{d}{2\pi e}}(\det(\mathcal{L}(B)))^{1/d}.$$

While it is trivial to construct lattices that do not follow the Gaussian Heuristic, it is frequently a useful tool for predicting the successive minima of lattices.

Lattice reduction algorithms input a basis B and output a basis B' for the same lattice with vectors satisfying some reducedness definition, typically ensuring that the output basis vectors are bounded and closer to orthogonal. The LLL algorithm [19] runs in polynomial time and returns [22, Theorem 9] a basis B' satisfying

$$\|\vec{b'}_1\|_2 \leq \alpha^{(d-1)/4} \det(\mathcal{L}(B))^{1/d}$$
$$\|\vec{b'}_i\|_2 \leq \alpha^{(d-1)/2} \lambda_i(\mathcal{L}(B))$$

for some $\alpha^{1/4} > 1.07$. That is, the lengths of the vectors are exponentially close to their optimal values. For random lattices, LLL lattice reduction achieves an even better approximation factor $\alpha^{1/4} \approx 1.02$ [23], so in typical cryptographic applications the approximation is very close or exact for small dimension lattices.

A surprisingly large number of cryptanalysis problems can be expressed in terms of finding short vectors in a lattice, and lattice reduction is a powerful tool for solving these problems. One example is an approach by Coppersmith [7] and later reinterpreted by Howgrave-Graham [14] to find small solutions to polynomials modulo integers. Coppersmith techniques are powerful, but Coppersmith lattices frequently have large dimension and large entries, so lattice reduction is expensive. In the case of multivariate polynomials, Coppersmith's method involves additional heuristic assumptions and a sometimes expensive Gröbner basis calculation [3,17].

2.2 The Hidden Number Problem

The Hidden Number Problem (HNP), introduced by Boneh and Venkatesan [6], poses the problem of recovering a hidden integer x from knowledge of a modulus N, multipliers t_i, and approximations $a_i = t_i x + e_i \pmod N$. In most presentations, these approximations a_i are given by an oracle that outputs the most significant bits of $t_i x \pmod N$, but we make the error term e_i more explicit in our generalization.

Boneh and Venkatesan gave an efficient lattice-based algorithm to solve this problem, and Bleichenbacher gave a Fourier analysis-based approach [8]. It is also possible to recover x from knowledge of the least significant bits of $t_i x \pmod N$ or some combination of the most and least significant bits [13].

A number of variants of HNP have been proposed in the literature. The Extended Hidden Number Problem (EHNP) considers the case when there are multiple contiguous blocks of unknown bits in $t_i x \pmod N$. This was defined by Hlaváč and Rosa [12] who also gave a lattice-based algorithm to solve it. EHNP is used as part of our cryptanalysis of MEGA and is discussed further in Sect. 4.4. The Hidden Number Problem with Chosen Multipliers (HNP-CM) considers t_i chosen adversarially instead of sampled at random [5], and can be solved without using lattice techniques via the ACGS algorithm [1]. In the Modular Inversion Hidden Number Problem (MIHNP), one wishes to recover a hidden number x from t_i and the most significant bits of $(x + t_i)^{-1} \pmod N$, and it can be solved via Coppersmith techniques [4]. The Hidden Number Problem with Hidden Multipliers (HNP-HM) recovers x from knowledge of the most significant bits of x, t_i, and $t_i x \pmod N$, and it can be solved using lattice techniques [16].

Our definition of HNP-SUM is similar to HNP-HM, but there are differences that prevent us from applying HNP-HM. First, HNP-HM requires uniformly distributed t_i, which our small unknown multipliers do not satisfy. HNP-HM also assumes the same number of bits of x and $t_i x \pmod N$ are known, whereas in our case the bounds T and E may not be equal. Finally, our goal in HNP-SUM is to recover the multipliers t_i and not the hidden number x. This is because for many parameters, the hidden number may not be unique: if x satisfies $a_i \approx t_i x \pmod N$, then it is likely $x + 1$ satisfies $a_i \approx t_i(x + 1) \pmod N$ as well.

3 Solving HNP-SUM

3.1 Solving HNP-SUM with $n = 2$

Before we solve HNP-SUM in the general case, we consider the case where $n = 2$. We are given N, a_1, a_2, T, and E satisfying

$$\begin{aligned} a_1 &\equiv t_1 x + e_1 \pmod N \\ a_2 &\equiv t_2 x + e_2 \pmod N \\ |t_1|, |t_2| &\le T \\ |e_1|, |e_2| &\le E. \end{aligned}$$

First we observe that the following linear expression is small modulo N. This is analogous to an observation by Faugère et al. [9] with an additional modular reduction in our setting.

$$\begin{aligned} t_2a_1 - t_1a_2 &\equiv t_2(t_1x + e_1) - t_1(t_2x + e_2) && \pmod N \\ &\equiv t_2e_1 - t_1e_2 && \pmod N \end{aligned}$$

Since both the t_i and e_i are bounded, this defines a linear system $ya_1 + za_2 \pmod N$ that has a small output $t_2e_1 - t_1e_2$ when evaluated at a small point $(t_2, -t_1)$. In order to find this small solution, we can look for a small vector in the lattice spanned by the rows of the following basis:

$$B = \begin{bmatrix} E & 0 & a_1 \\ 0 & E & a_2 \\ 0 & 0 & N \end{bmatrix}$$

The vector $\vec{v} = (Et_2, -Et_1, t_1e_2 - t_2e_1)$ is small and is contained in this lattice, so we might hope that lattice reduction will find it.

However, there might be a small obstruction if t_1 and t_2 are not relatively prime. Note that if t_1 and t_2 share a common factor g, then the vector $\vec{v}/g = (Et_2/g, -Et_1/g, t_1e_2/g - t_2e_1/g)$ is also in the lattice, and it is shorter than $\vec{v}$. We observe experimentally that lattice reduction typically outputs one of the vectors $\pm\vec{v}/g$. For our definition of HNP-SUM, we only require finding t_i up to sign and common factor, but we the issue of common factors also appears for $n > 2$ and requires more analysis to work around.

Theorem 2. *HNP-SUM defined as in Definition 1 with two samples is solvable in heuristic polynomial time when*

$$T^3E \lesssim N.$$

Proof. The Gaussian Heuristic predicts that $\lambda_1(\mathcal{L}(B)) \approx (E^2N)^{1/3}$, and we also have $\lambda_1(\mathcal{L}(B)) \leq \|\vec{v}/g\|_2 \approx ET$. If $T^3E \lesssim N$, then $ET \lesssim (E^2N)^{1/3}$ and the Gaussian Heuristic is invalid. Instead of behaving like a random lattice, $\mathcal{L}(B)$ contains an unexpectedly short vector, which is heuristically the shortest vector in the lattice. If $\vec{v}/g$ is the shortest vector by some small constant factor, then the LLL algorithm applied to B finds $\pm\vec{v}/g$, which reveals (t_1, t_2) up to sign and common factor.

3.2 Construction for $n > 2$

The approach for solving HNP-SUM with $n > 2$ is a natural extension of the previous section. Inspired by the construction for $n = 2$, we study the lattice spanned by the rows of the basis matrix

$$B = \begin{bmatrix} E & & & & a_1 \\ & E & & & a_2 \\ & & \ddots & & \vdots \\ & & & E & a_n \\ & & & & N \end{bmatrix}$$

where all unspecified entries are 0. The rank of the lattice is $n + 1$ and the determinant is $E^n N$. In the $n = 2$ case, lattice reduction can be used to find $t_2, -t_1$ such that $t_2 a_1 - t_1 a_2 \equiv t_2 e_1 - t_1 e_2 \pmod{N}$ is small. For $n > 2$, we have many such small linear combinations from each pair (a_i, a_j) and combinations of these pairs.

Unlike the $n = 2$ case, reducing this lattice does not result in a single short vector. Instead, we empirically observe that lattice reduction returns $n - 1$ linearly independent short vectors of the same length, corresponding to a dense sublattice of rank $n - 1$. This sublattice is related to the (t_i, t_j) pairs and must be postprocessed to recover the individual unknown multipliers t_i.

More concretely, we consider the sublattice $\mathcal{L}(B_{sub})$ containing the set of (linearly dependent) short vectors

$$\begin{aligned}\vec{v}_{i,j} &= t_j \vec{b}_i / \gcd(t_i, t_j) + t_i \vec{b}_j / \gcd(t_i, t_j) - k_{i,j} \vec{b}_{n+1} \\ B_{sub} &= \{\vec{v}_{i,j} \mid i, j \in \{1, \dots, n\}, i \neq j\}\end{aligned}$$

where $\vec{b}_i$ is the i^{th} row vector of B. Vector $\vec{b}_{n+1}$ is included to achieve modular reduction by N, and $k_{i,j}$ is set so the last entry of $\vec{v}_{i,j}$ is $(t_j e_i - t_i e_j)/\gcd(t_i, t_j)$. With this definition, B_{sub} contains the short vectors

$$\vec{v}_{i,j} = (0, \dots, E t_j, \dots, -E t_i, \dots, 0, t_j e_i - t_i e_j)/\gcd(t_i, t_j).$$

Clearly $\mathcal{L}(B_{sub})$ is a sublattice of $\mathcal{L}(B)$, but it is not obvious what the rank or determinant of $\mathcal{L}(B_{sub})$ is or how to recover the t_i from knowledge of this particular sublattice.

Section 3.3 explores an alternative basis H for this sublattice that gives insight into its structure. Section 3.4 shows how to recover the unknown multipliers from a basis of the sublattice by computing the Hermite Normal Form of the basis. Section 3.5 bounds the determinant of this sublattice, and finally Sect. 3.6 gives heuristic bounds for when lattice reduction can be used to find a basis of this sublattice.

3.3 Alternative Basis for the Sublattice

We begin by constructing matrix $H \in \mathbb{Z}^{n-1 \times n+1}$, which we show is a basis for $\mathcal{L}(B_{sub})$. Although H is not the HNF of the lattice, it is closely related. We define the rows of H by

$$\vec{h}_i = \begin{cases} \sum_{j=i+1}^{n} u_{i,j} \vec{v}_{i,j} & \text{for } i < n-1 \\ \vec{v}_{i,i+1} & \text{for } i = n-1 \end{cases}$$

where $u_{i,j}$ are integer values found by the extended GCD algorithm such that $\sum_{j=i+1}^{n} u_{i,j} \frac{t_j}{\gcd(t_i, t_j)} = \gcd(\frac{t_{i+1}}{\gcd(t_i, t_{i+1})}, \dots, \frac{t_n}{\gcd(t_i, t_n)}) = \frac{\gcd(t_{i+1}, \dots, t_n)}{\gcd(t_i, \dots, t_n)} = \tilde{g}_i$. This gives H the structure

$$H = \begin{bmatrix} \tilde{g}_1 E & * & \dots & * & * & * & * \\ & \tilde{g}_2 E & \dots & * & * & * & * \\ & & \ddots & \vdots & \vdots & \vdots & \vdots \\ & & & \tilde{g}_{n-2} E & * & * & * \\ & & & & \frac{t_n E}{\gcd(t_{n-1},t_n)} & \frac{-t_{n-1} E}{\gcd(t_{n-1},t_n)} & \frac{t_n e_{n-1} - t_{n-1} e_n}{\gcd(t_{n-1},t_n)} \end{bmatrix}$$

where the entries below the diagonal are zero.

Lemma 1. *H is a basis for the sublattice $\mathcal{L}(B_{sub})$. That is, $\mathcal{L}(B_{sub}) = \mathcal{L}(H)$.*

The construction of H makes it clear that $\vec{h}_i \in \mathcal{L}(B_{sub})$, and it is straightforward but tedious to show inclusion in the other direction. We include a proof of Lemma 1 in the full version of this paper [26].

Like the HNF, H is upper triangular, and the elements along the diagonal of H reveal the pivots of the HNF. Although the entries above the diagonal may not be in the necessary range for the HNF, that is easily fixed by doing simple row operations that do not modify the values along the diagonal.

3.4 Recovering Unknown Multipliers

At this point, we assume that we have applied a lattice reduction algorithm to B and have obtained a basis B'_{sub} for $\mathcal{L}(B_{sub})$. We compute $H' = \text{HNF}(B'_{sub}) = \text{HNF}(H)$ in polynomial time and learn the pivots $\tilde{g}_1 E, \dots, \tilde{g}_{n-2} E$ as well as the values $\frac{\pm t_n E}{\gcd(t_{n-1},t_n)}, \frac{\mp t_{n-1} E}{\gcd(t_{n-1},t_n)}$.

Setting $G = \gcd(t_1, \dots, t_n)$, the definition of $\tilde{g}_i$ shows that the product of the these are $\prod_{i=1}^{n-2} \tilde{g}_i = \frac{\gcd(t_{n-1},t_n)}{G}$, so knowledge of the HNF allows us to compute $\left(\prod_{i=1}^{n-2} \tilde{g}_i\right) \frac{\pm t_n}{\gcd(t_{n-1},t_n)} = \pm t_n / G$. Similarly, we can compute $\mp t_{n-1}/G$, revealing the pair (t_{n-1}, t_n) up to sign and division by G.

Note that the ordering of the samples a_i is arbitrary, and by reordering and repeating the process, we can learn any pair (t_i, t_j) up to sign and division by G. Rather than performing multiple lattice reductions, we can just reorder the columns of B'_{sub} and recompute the HNF for each (t_i, t_j) pair we wish to recover. By recovering $(t_1, t_n), (t_2, t_n), \dots, (t_{n-1}, t_n)$ up to sign and division by G, we learn $\vec{t'} = \pm(t_1, t_2, \dots, t_n)/G$, which is $\vec{t}$ up to sign and division. This is a valid solution for HNP-SUM.

3.5 Sublattice Determinant

The previous sections show the existence of a sublattice $\mathcal{L}(B_{sub})$ of rank $n - 1$. Experimentally, we observe that lattice reduction finds this sublattice. Our goal is to heuristically understand the properties of the sublattice in order to characterize when we expect our lattice attack to succeed. We make the following heuristic observation by calculating the sublattice basis H for random instances of HNP-SUM and computing the determinant.

Heuristic 1. Let $N, a_i, T, E, t_i, e_i, x$ define an instance of HNP-SUM with x, t_i, e_i drawn uniformly at random from their respective ranges. Let $\mathcal{L}(B_{sub})$ be the sublattice defined in Sect. 3.3. Then experimentally

$$\det(\mathcal{L}(B_{sub})) \approx 0.35nE^{n-1}T.$$

While the heuristic suffices to predict the behavior of our method on random instances of HNP-SUM, we can also prove a weaker version of our heuristic for a restricted class of HNP-SUM instances. Note that although this analytic bound is slightly worse, it has the same structure as our heuristic, providing further evidence for the claims of our heuristic approach.

Lemma 2. *Let the rows of $H \in \mathbb{Z}^{n-1 \times n+1}$ define a basis for the sublattice $\mathcal{L}(B_{sub})$ associated with an instance (N, a_i, T, E) of HNP-SUM where the two values of t_i with the largest magnitude are coprime.*

The determinant of the sublattice is bounded:

$$\det(\mathcal{L}(B_{sub})) < n(2E)^{n-1}T.$$

The proof for this lemma considers the special structure of H which arises from the coprimality of the two large t_i and is included in the full version [26].

3.6 Sublattice Recovery via Lattice Reduction

In the previous sections, we demonstrated the existence of a sublattice $\mathcal{L}(B_{sub})$ of rank $n-1$ with heuristic determinant approximately $nE^{n-1}T$. It remains to establish the conditions under which lattice reduction finds this sublattice. LLL lattice reduction on B finds short vectors that approximate the successive minima λ_i of $\mathcal{L}(B)$. To show that lattice reduction finds the sublattice of rank $n-1$, we first estimate the successive minima, and then we argue that the first $n-1$ vectors found by lattice reduction must belong to the sublattice and therefore form a basis.

Since the determinant of $\mathcal{L}(B)$ depends on N and E and there exists a dense sublattice with determinant that depends on T and E, the Gaussian Heuristic does not hold in general for our lattice. However, we make the following heuristic assumption which leads to accurate bounds for our method.

Heuristic 2. Let B be the basis of rank $n+1$ for an instance of HNP-SUM. Let $\mathcal{L}(B_{sub})$ be the sublattice specified in Sect. 3.2 and let $\mathcal{L}(B_{\perp})$ be the rank-2 lattice formed by projecting $\mathcal{L}(B)$ orthogonally to the span of B_{sub}. We assume the successive minima of $\mathcal{L}(B_{sub})$ and the successive minima of $\mathcal{L}(B_{\perp})$ each follow the Gaussian Heuristic.

We can use this heuristic to infer the successive minima of $\mathcal{L}(B)$. A proof of Lemma 3 is in the full version [26].

Lemma 3. *Let N, n, T, E be parameters for an instance of HNP-SUM where $n^{2n/(n-1)}ET^{(n+1)/(n-1)} \lesssim N$. Let B the constructed lattice basis, and assume Heuristic 1 and Heuristic 2 hold. Then the successive minima of $\mathcal{L}(B)$ follow*

$$\lambda_i(\mathcal{L}(B)) \approx \begin{cases} n^{(n+1)/(2n-2)}ET^{1/(n-1)} & 1 \le i \le n-1 \\ \sqrt{\frac{NE}{nT}} & n \le i \le n+1 \end{cases}$$

up to a small constant factor, and the vectors corresponding to the first $n-1$ minima are in $\mathcal{L}(B_{sub})$.

If the rank of the lattice is small enough that we can recover the shortest vectors exactly, then this reveals a basis for $\mathcal{L}(B_{sub})$, and Sect. 3.4 shows how to recover the unknown multipliers. If the rank of the lattice is too large, we can use LLL lattice reduction to find a basis for the sublattice. Proving this is straightforward given Lemma 3.

Lemma 4. *Let N, n, T, E be parameters for an instance of HNP-SUM where $\alpha^n n^{2n/(n-1)}ET^{(n+1)/(n-1)} \lesssim N$ for some fixed approximation factor $\alpha > 4/3$, and assume Heuristics 1 and 2 hold. Let $\mathcal{L}(B_{sub})$ be the sublattice as before. It is possible to recover a basis for $\mathcal{L}(B_{sub})$ in polynomial time.*

For small to medium values of n, the LLL approximation factor α and the term $n^{2n/(n-1)}$ change the exact bounds by only a few bits, so for most cases in practice, it suffices to use the heuristic $ET^{(n+1)/(n-1)} \lesssim N$. Combining Lemma 4 with the method in Sect. 3.4 leads to a proof of Theorem 1.

3.7 Experimental Evaluation

We implemented our algorithm for solving HNP-SUM using Python and SageMath. We use SageMath to compute the Hermite Normal Form and a custom C++ implementation to perform lattice reduction. We experimentally measured the success rate of our algorithm for various parameters. We randomly generated 2048-bit moduli and, depending on our choice of n, E, and T, we generated t_i and e_i uniformly at random to construct the HNP-SUM samples a_i. Our experiments reduced lattices of dimension up to 31 and entries of size 2048 bits, and lattice reduction took under a half second to complete on a single thread of an Intel Xeon E5-2699A processor running at 2.4 GHz. Our results in Fig. 1 show that our predicted bound $T^{(n+1)/(n-1)}E \lesssim N$ is accurate to within a few bits in practice.

4 Application: Cryptanalyzing MEGA

We present two novel and overlapping key recovery attacks on MEGA. The first (*fast*) attack requires as few as 16 login attempts (with 17 login attempts on average) and takes only a few seconds to complete. The second (*small*) attack requires only 6 login attempts to succeed with 98% probability, but it is more

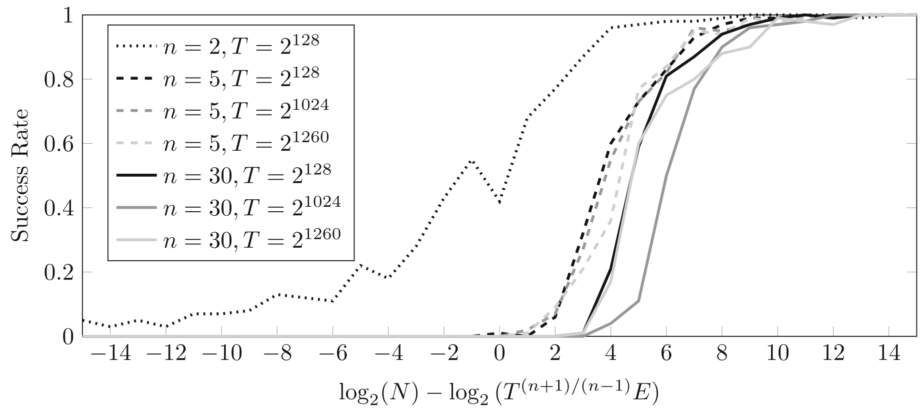

Fig. 1. Success Rate of our HNP-SUM method. We generated random instances of HNP-SUM with $N \approx 2^{2048}$, $n \in \{2, 5, 30\}$, and $T \in \{2^{128}, 2^{1024}, 2^{1260}\}$. We set E to be close to the threshold value predicted by our bound $T^{(n+1)/(n-1)}E \lesssim N$, skipping the cases $n = 2, T \geq 2^{1024}$ for which no E satisfies the bound. Each data point is averaged from 100 sample instances. In all cases, the actual threshold is within a small factor of the predicted threshold, showing that our heuristic assumptions are valid. We see that the threshold is slightly higher for larger n, suggesting that the bounds likely have a secondary dependence on n as well.

computationally intensive. This latter attack can be performed in 4.5 h on an 88-core machine, but we include both because the former can be easily verified and includes some interesting additional analysis steps. Both of these attacks proceed in roughly the same series of stages with only minor variations in how the stage is completed in both the fast attack and the small attack. While solving HNP-SUM is a stage in both attacks, a number of additional stages are needed to preprocess and postprocess the leakage from the client to get it into the correct form. As a motivating application of HNP-SUM, we describe the relevant details of MEGA's protocol and the sequence of stages that allows for more efficient key recovery.

In MEGA's login protocol, the server sends the client an RSA private key that is encrypted using AES in ECB mode. The client decrypts the RSA private key, uses this RSA private key to decrypt a session ID that the server has encrypted to the RSA public key, and sends the result to the server.

The attack of Backendal, Haller, and Paterson modifies the ECB-encrypted ciphertext of the RSA private key and the encrypted session ID to obtain one bit of information about the secret key per login. However, the client is using the modified secret key to send 43 contiguous bytes of information from the result of the RSA decryption to the server. In our attack, the adversary swaps blocks in the ECB-encrypted wrapped RSA key before sending it to the client and then analyzes the resulting message from the client to obtain more information about the RSA secret key per victim client login attempt.

In the first stage of analysis, the attacker represents the 43-byte leakage from the client in terms of the unknown AES plaintext blocks. Second, these algebraic representations are manipulated so that the attacker learns information about the most significant bytes (MSBs) of an unknown value, not just about a contiguous subsequence of bytes. In the fast attack, this is done using an approach from solutions to the Extended Hidden Number Problem [12], and in the small attack, this is done by brute forcing unknown most significant bytes. Third, in the fast attack, these approximations of the MSBs are refined by combining approximations together so more MSBs are known. This is why the fast attack requires more samples than the small attack. Fourth, the (refined) approximations are used to solve for the value of unknown multipliers in the algebraic representation via HNP-SUM. These unknown multipliers correspond to differences between plaintext blocks in the encoded RSA private key. Fifth, we use the RSA equations to brute force a block of plaintext bytes of the RSA private exponent in the encoded key, and the plaintext differences reveal the values of other plaintext blocks. Finally, the plaintext blocks containing the MSBs of one of the RSA factors are analyzed in a Coppersmith attack [7] to recover the full factorization and private key.

Section 4.3 through 4.8 discuss each of the stages in turn. Section 4.9 analyzes the overall complexity of the attack.

4.1 Attack Context for MEGA

When a MEGA user attempts to log in for the first time on a new client, the client is only in possession of an AES secret key derived from the user's password. To function properly, the client requires a copy of the user's RSA private key. The server possesses the user's RSA public key and a copy of the user's RSA private key encrypted using the AES key in ECB mode, so in theory the private key is hidden from the server, but the client can obtain the private RSA key by decrypting the encrypted private key from the server with the password-derived AES secret key. It is the malicious server's goal is to recover the user's private RSA key.

During the login process, the server creates a 43-byte session identifier (SID), which it encrypts using the RSA public key and sends to the client alongside the wrapped private key. The client uses the AES key to unwrap the RSA private key, then uses the parameters in the unwrapped key to decrypt the RSA ciphertext and retrieve the SID. The client then sends the retrieved SID to the server. The malicious server wishes to use the SID value sent from the client to infer information about the parameters in the unwrapped private key.

Several of the exact implementation details are relevant for our improved attack, so we recount them here. The remaining details can be found in Backendal, Haller, and Paterson's paper [2, Section II]. We denote the RSA public key by (N, e), where the factors of modulus N are p and q. The public RSA exponent is e and the private exponent is d. The public RSA exponent is set by the client; the web client[1] uses $e = 257$, and the SDK[2] uses $e = 17$. MEGA clients use RSA-CRT for decryption, so let $u \leftarrow q^{-1} \bmod p$ be the coefficient used during CRT operations.

The private key is encoded as

$$sk_{share}^{encoded} \leftarrow \ell(q) \mid q \mid \ell(p) \mid p \mid \ell(d) \mid d \mid \ell(u) \mid u \mid P.$$

l encodes the bit length of different values as a 2-byte integer, all integers are stored in big-endian format, and P is an 8-byte padding value unknown to the adversary. We wish to highlight that q is 1024 bits in length, so $\ell(q)$ is `0x0400`, and since the secret values are of predictable size, they appear at predictable offsets within the plaintext. We also highlight that the private key encodes the full private exponent d and does not include the private exponents d_p, d_q that are frequently stored for use in RSA-CRT decryption. Finally, we note that due to the length fields, the 1024-bit u value spans 9 AES plaintext blocks, and the first and last of those contain the length and padding fields respectively. As in the original attack, we constrain our attacker to not alter this metadata so that the decrypted RSA key maintains correct length encodings and padding.

This encoding of the private key is 656 bytes, or 41 AES blocks. The encoded private key is encrypted using AES in ECB mode, which means that each 16-byte plaintext block is encrypted independently. That is,

$$ct_1 \mid ct_2 \mid \cdots \mid ct_{41} = \mathrm{E}_{\mathrm{AES}}(pt_1) \mid \mathrm{E}_{\mathrm{AES}}(pt_2) \mid \cdots \mid \mathrm{E}_{\mathrm{AES}}(pt_{41}).$$

Decryption of the encrypted private key also processes 16-byte blocks independently, enabling malleability attacks where the malicious server alters individual blocks of ciphertext to alter the corresponding blocks of plaintext in the private key encoding.

When the honest server constructs the RSA plaintext with the 43-byte SID, it places the SID in bytes 3-45 of the 256-byte RSA plaintext m. Prior to patching, clients extract these bytes from the RSA decryption output without checking the validity of the remainder of the decryption output. However, there is special behavior in the client's extraction function that checks if byte 2 is nonzero, and if this is the case it extracts bytes 2-44. This detail has no consequence for the RSA key extraction attack in [2], but it is a necessary aspect of our small attack. If we assume the output bytes of the RSA decryption function are uniformly distributed, clients have probability 255/256 of returning $SID \leftarrow m[2:44]$. We temporarily set this detail aside and assume that all SIDs returned by the client are composed of these bytes, and we revisit it in Sect. 4.9.

MEGA clients use Garner's formula [11] to perform RSA-CRT decryption, the process of decrypting an RSA ciphertext c to message m. These equations, as well as the SID extraction step, are detailed below.

1 https://github.com/meganz/webclient/blob/9fca1d0b7d8a65b9d483a10e798f1f67d1fb8f1e/js/crypto.js#L207.

2 https://github.com/meganz/sdk/blob/849aea4d49e2bf24e06d1a3451823e02abd76f39/src/crypto/cryptopp.cpp#L798.

$$\begin{aligned} m_p &\leftarrow c^{d \bmod (p-1)} \bmod p \\ m_q &\leftarrow c^{d \bmod (q-1)} \bmod q \\ m &\leftarrow ((m_p - m_q)u \bmod p)q + m_q \\ SID &\leftarrow m[2:44] \end{aligned}$$

4.2 Original MEGA Attack of Backendal, Haller, and Paterson

In the original attack, the adversary alters ciphertext block ct_{40}, which is the last ciphertext block corresponding to only bytes of u, and no length fields or padding bytes. The attacker sends this altered wrapped key and RSA ciphertext $q_{guess}^e \bmod N$ to the client. The client decrypts and decodes the wrapped key to obtain private key (q, p, d, u', P) where $u' \neq u$ is not the correct value of $q^{-1} \bmod p$ to use during RSA-CRT decryption.

If $q_{guess} < q$, then $m_p \equiv m_q \pmod p$, so $(m_p - m_q)u = 0$ and $m = m_q < q$. Thus all SID bytes are 0. If $q_{guess} \geq q$, then $m_p \not\equiv m_q \pmod p$, so $h \neq 0$ and $m > q$. Thus the SID bytes are nonzero with high probability. The attack therefore uses whether SID is zero or nonzero as an oracle for whether the attacker-chosen q_{guess} is smaller or larger than the secret q. The adversary does a binary search on the value of q until sufficiently many most significant bits of q are known.

Once enough of the most significant bits of q are known, the attacker then uses a cryptanalytic technique by Coppersmith [7] to recover the least significant bits of q, and thus obtain the full factorization of N. Asymptotically, this attack recovers the factorization of N in polynomial time once the attacker knows the most significant half of bits of q. In the context of MEGA, that is 512 bits, which requires 512 login attempts to obtain. In practice, this attack can be prohibitively slow at the asymptotic limit, and implementations of Coppersmith's method often use additional most significant bits, which makes the implementation faster and more understandable. The proof-of-concept code associated with the original attack uses 683 most significant bits and therefore requires 683 login attempts.

We observe that although the client provides the adversary with 344 bits of SID per login attempt, this original attack only uses this data to refine the knowledge of the private key by a single bit. It is natural to wonder if the client's responses can be exploited in a more sample-efficient way, recovering the same private key with fewer login attempts. This is what our new attacks accomplish.

4.3 Expressing Leakage Algebraically

We begin our cryptanalysis by demonstrating how to algebraically express the information returned during a login attempt. As in the original attack, the adversary alters ciphertext blocks corresponding to the value of u, and therefore the client uses the altered u' value when performing decryption, but the remaining private key values are unaltered. In our attack, the adversary also picks an RSA ciphertext c at random, and reuses the same c for each login attempt. Both the (modified) wrapped key and RSA ciphertext are sent to the client during a login attempt.

By combining Garner's formula for RSA decryption with the extraction of the SID s' with altered value u', this gives the congruence

$$(m_p - m_q)u'q + m_q \equiv e_1'2^{b_1} + s'2^{b_2} + 2^{b_2-1} + e_2' \pmod{N}.$$

The left hand side expresses the output of the decryption function in terms of its input, and the right hand side expresses the output in terms of the known SID bytes $s' = m'[2:44]$ and the other unknown bytes $e_1' = m'[1]$ and $e_2' = m'[45:256] - 2^{b_2-1}$. The 2^{b_2-1} term is present so that unknown e_2' may be positive or negative and so $|e_2'|$ is minimized.

We can construct a similar equation using altered value u'' and SID s''.

$$(m_p - m_q)u''q + m_q \equiv e_1''2^{b_1} + s''2^{b_2} + 2^{b_2-1} + e_2'' \pmod{N}.$$

Subtracting these two congruences, we have

$$(u' - u'')(m_p - m_q)q \equiv (e_1' - e_1'')2^{b_1} + (s' - s'')2^{b_2} + (e_2' - e_2'') \pmod{N}.$$

The adversary can give extra structure to $(u' - u'')$ by carefully manipulating the AES-encrypted key. The value u used by the client during RSA decryption is decoded from the nine AES-decrypted blocks $\mathrm{D_{AES}}(ct_{33} \mid ct_{34} \mid \cdots \mid ct_{41})$. Plaintext blocks pt_{33} and pt_{41} also include some bytes of d, the encoding of $\ell(u)$, and padding P. Now observe that if the attacker swaps out some of these ciphertext blocks encrypting u with ciphertext blocks ct_i, ct_j of their choosing, the decrypted and decoded value of u used by the client will contain bits from pt_i and pt_j. Consider what happens when the client decodes u' and u'' from the following two ciphertexts, which differ in the second-to-last block:

$$\begin{aligned} u' &= \mathrm{Decode}[\mathrm{D_{AES}}(ct_{33}) \mid \mathrm{D_{AES}}(ct_i) \mid \cdots \mid \mathrm{D_{AES}}(ct_i) \mid \mathrm{D_{AES}}(ct_i) \mid \mathrm{D_{AES}}(ct_{41})] \\ u'' &= \mathrm{Decode}[\mathrm{D_{AES}}(ct_{33}) \mid \mathrm{D_{AES}}(ct_i) \mid \cdots \mid \mathrm{D_{AES}}(ct_i) \mid \mathrm{D_{AES}}(ct_j) \mid \mathrm{D_{AES}}(ct_{41})], \end{aligned}$$

After decryption, all of the plaintext blocks that contain only bits of u are replaced with pt_i, except for one in the second plaintext which is replaced with pt_j. The plaintext blocks that contain length encoding data or padding are not modified, so validation of the plaintext succeeds. With this construction, $(u' - u'')$ has special structure, because the only difference between the two is in block 40, which corresponds to bytes 105 through 120 of the encoded u. Therefore,

$$u' - u'' = (pt_i - pt_j)2^{64}.$$

For simplicity, in the future we will denote $\delta_{i,j} = pt_i - pt_j$, and observe that $|\delta_{i,j}| < 2^{128}$.

We will also consider $u' - u'''$ when u''' was decoded from the ciphertext

$$u''' = \mathrm{Decode}[\mathrm{D_{AES}}(ct_{33}) \mid \mathrm{D_{AES}}(ct_i) \mid \cdots \mid \mathrm{D_{AES}}(ct_j) \mid \mathrm{D_{AES}}(ct_i) \mid \mathrm{D_{AES}}(ct_{41})]$$

which differs only in block 39. By the same logic as before,

$$u' - u''' = (pt_i - pt_j)2^{196} = 2^{128}\delta_{i,j}2^{64}.$$

This generalizes so that the adversary can construct values of u with difference $\delta_{i,j}2^{128t+64}$ for $t \in \{0, 1, \ldots, 6\}$, corresponding to the 7 modifiable ciphertext blocks that contain only bytes of u and no padding bytes.

4.4 Obtaining Most Significant Bytes

For any AES ciphertext block indices i and j, Sect. 4.3 gives us the capability to construct an equation involving the differences of the corresponding plaintexts $\delta_{i,j} = pt_i - pt_j$. Specifically, we have

$$\delta_{i,j} 2^{128t+64}(m_p - m_q)q \equiv (e_1' - e_1'')2^{b_1} + (s' - s'')2^{b_2} + (e_2' - e_2'') \pmod{N}.$$

In this equation, the adversary knows $(s' - s'')$ because it is the difference of two SIDs, and the adversary also knows t, b_1, b_2, and N. The adversary does not know $2^{64}(m_p - m_q)q \bmod N$, but this value is constant throughout the attack. The adversary does not know $(e_1' - e_1'')$ or $(e_2' - e_2'')$, but knows they are bounded by $|e_1' - e_1''| \le E_1 = 2^8$ and $|e_2' - e_2''| \le E_2 = 2^{b_2}$.

The goal of this phase is to learn the most significant bytes of some algebraic expression. This is a generally useful goal because it allows us to represent the error in the approximation as some bounded variable, and it is frequently possible to efficiently solve the problem of recovering bounded variables using lattice methods.

We now detail two approaches for obtaining the most significant bytes of this representation.

Brute Force. Because e_1' and e_1'' are both single-byte values, $e_1' - e_1''$ takes on one of 511 values. We can brute force these values and expect to eventually guess the correct value. Therefore, assuming we have guessed correctly, we can compute $a = (e_1' - e_1'')2^{b_1} + (s' - s'')2^{b_2}$ and write

$$2^{128t}\delta_{i,j}x \equiv a - \varepsilon \pmod{N}$$

where $x = 2^{64}(m_p - m_q)q \bmod N$ is unknown but constant throughout the attack. This is beginning to resemble a sample for an instance of HNP-SUM with unknown multiplier $2^{128t}\delta_{i,j}$ and error ε which is unknown and bounded by $|\varepsilon| \le 2^{b_2} = 2^{1696}$.

Extended Hidden Number Problem. We observe that the problem of converting a sample with a known block of contiguous bytes into a sample with known most significant bytes (MSBs) resembles the Extended Hidden Number Problem (EHNP) [12], specifically the Hidden Number Problem with two holes (HNP-2H). To obtain the MSBs, we search for a known multiplier C which simultaneously makes the unknown terms $(e_1' - e_1'')C2^{b_1} \bmod N$ and $(e_2' - e_2'')C \bmod N$ small. If we assume $|e_1' - e_1''| < E_1$ and $|e_2' - e_2''| < E_2$, such a value of C can be found by reducing the lattice defined by the rows of the basis matrix $B =$

$$\begin{bmatrix} E_1 N & 0 \\ E_1 2^{b_1} & E_2 \end{bmatrix}.$$

Lattice reduction finds the shortest vector $v = (E_1(C2_1^b \bmod N), E_2C)$ with $\|v\|_2 \leq \frac{2}{\sqrt{3}}\det \mathrm{B}^{1/2} = \frac{2}{\sqrt{3}}\sqrt{E_1E_2N}$. Thus

$$\begin{aligned}
&|(e_1' - e_1'')C2^{b_1} + (e_2' - e_2'')C \bmod N| \\
\leq &|e_1' - e_1''||C2^{b_1} \bmod N| + |e_2' - e_2''||C| \\
\leq &E_1|C2^{b_1} \bmod N| + E_2|C| \\
\leq &\|v\|_2 + \|v\|_2 \\
\leq &\frac{4}{\sqrt{3}}\sqrt{E_1E_2N}.
\end{aligned}$$

We set $C = v_2/E_2$ and note that C does not depend on information leaked from the client, and thus can be reused for every sample.

We therefore let $x = C(m_p - m_q)q \bmod N$, $a = C(s' - s'')2^{b_2}$, and $\varepsilon = -(e_1' - e_1'')C2^{b_1} - (e_2' - e_2'')C \bmod N$. This yields

$$2^{128t}\delta_{i,j}x \equiv a - \varepsilon \pmod N.$$

This also resembles a sample for an instance of HNP-SUM with unknown multiplier $2^{128t}\delta_{i,j}$, known approximation a that depends on the SIDs and C, and error ε which is unknown and bounded by $|\varepsilon| \leq \frac{4}{\sqrt{3}}\sqrt{E_1E_2N} \leq 2^{1878}$.

The approach using the EHNP technique therefore produces a similar equation to the brute-force approach, but the bound on the unknown ε is larger. In fact, this approach loses about half of the information exposed by the client; instead of knowing 43 MSBs, this transformation gives information about only 21.25 MSBs.

4.5 Refining Approximations

Our ability to solve HNP-SUM depends on the bounds for the multiplier and the error, and the error in the HNP-SUM samples we can obtain via the EHNP method is too large to be recovered. When using the EHNP method, it is therefore necessary to combine multiple HNP-SUM samples together to obtain a sample with smaller error. For the particular context of this attack, this sample refinement is possible.

Specifically, for any AES block indices i, j and choice of $t \in \{0, 1, \ldots, 6\}$, the adversary uses Sect. 4.4 to learn a_t satisfying

$$2^{128t}\delta_{i,j}x \equiv a_t - \varepsilon_t \pmod N.$$

$\delta_{i,j} = pt_i - pt_j$ is the difference of two plaintexts and is bounded $|\delta_{i,j}| \leq 2^{128}$. We also have bound $|\varepsilon_t| < E$. The goal of the adversary is to *refine* the approximation by computing $\tilde{a}$ satisfying

$$\delta_{i,j}x \equiv \tilde{a} - \tilde{\varepsilon} \pmod N$$

where $|\tilde{\varepsilon}| \leq \tilde{E} \leq E$.

Since the new bound on the error is smaller, this is equivalent to learning additional MSBs of $\delta_{i,j}x$.

We simplify the problem to a single refinement step using two approximations. Once we show that this is possible, it is clear that this can be repeated multiple times to refine the approximation further. We state the problem generically.

Approximation Refinement Problem. Assume the adversary is given $a_1, a_2, r \neq 0, N, E_1$ and E_2 satisfying

$$\begin{aligned} y &\equiv a_1 - \varepsilon_1 \pmod N \\ ry &\equiv a_2 - \varepsilon_2 \pmod N \\ |\varepsilon_1| &\leq E_1 \\ |\varepsilon_2| &\leq E_2 \\ 2|r|E_1 + 1 &\leq N - 2E_2. \end{aligned}$$

If $\min((2E_2+1)/|r|, 2E_1+1) < 2\tilde{E}$, then the attacker's goal is to return $\tilde{a}$ such that there exists $\tilde{\varepsilon}$ satisfying $|\tilde{\varepsilon}| \leq \tilde{E}$ and

$$y \equiv \tilde{a} + \tilde{\varepsilon} \pmod N.$$

Intuitively, we consider the intersection of the set of y values satisfying the first congruence with the set of y values satisfying the second congruence. Because of the constraints on the parameters, the intersection is a single interval with easily computed bounds.

To solve this problem, observe that there exists y satisfying $y \in [a_1 - E_1, a_1 + E_1]$. Without loss of generality, assume $r > 0$. so therefore $ry \in S_1 = [r(a_1 - E_1), r(a_1 + E_1)]$. Also observe that

$$ry \in S_2 = \bigcup_{k=-\infty}^{\infty} [a_2 - E_2 + kN, a_2 + E_2 + kN],$$

so we wish to find the intersection of S_1 and S_2. Because S_2 consists of the union of intervals of size $2E_2 + 1$, repeated at multiples of N, the gaps between these intervals are $N - 2E_2 - 1$. Since the size of S_1 is $2rE_1 + 1 \leq N - 2E_2$, S_1 intersects with at most one interval, and we know there exists ry, the intersection of S_1 and S_2 is a single interval. Therefore we compute

$$\begin{aligned} k^* &\leftarrow \left\lceil \frac{r(a_1 - E_1) - (a_2 + E_2)}{N} \right\rceil \\ low &\leftarrow \max(r(a_1 - E_1), a_2 - E_2 + k^*N) \\ high &\leftarrow \min(r(a_1 + E_1), a_2 + E_2 + k^*N) \end{aligned}$$

and observe

$$ry \in S_1 \cap S_2 = [low, high] \Rightarrow y \in \left[\left\lceil \frac{low}{r} \right\rceil, \left\lfloor \frac{high}{r} \right\rfloor \right].$$

The size of this interval is at most $\min((2E_2+1)/r, 2E_1+1) < 2\tilde{E}$, so we let $\tilde{a}$ be its midpoint (or as close as possible if there are an even number of elements) and we have solved the problem.

To apply this to our specific problem, observe that this means that we can refine the EHNP sample $\delta_{i,j}x \equiv a_0 - \varepsilon_0 \pmod{N}$ with $2^{128}\delta_{i,j}x \equiv a_1 - \varepsilon_1 \pmod{N}$ to quality $\tilde{E} = 2^{1750}$ because $r = 2^{128}$, $E_1 = E_2 = 2^{1878}$, $N \approx 2^{2048}$. Similar logic shows that we can iterate this process, using the three samples $\{a_0, a_1, a_2\}$ to obtain a refined sample of the form

$$\delta_{i,j}x \equiv \tilde{a} - \tilde{\varepsilon} \pmod{N} \text{ with } |\tilde{\varepsilon}| \leq 2^{1622}.$$

This increases the number of known MSBs from about 21 to 53 and produces an HNP-SUM sample with small enough error to enable finding a solution.

4.6 Recovering Unknown Multipliers

We now turn to the goal of recovering unknown and small multipliers. For arbitrarily many (i,j) pairs, the attacker knows $a_{i,j}$ such that

$$a_{i,j} \equiv \delta_{i,j}x + e_{i,j} \pmod{N}$$

where $|\delta_{i,j}| \leq T = 2^{128}$ and $|e_{i,j}| < E$. The value of E depends on if the adversary initially used the brute-force strategy (giving $E = 2^{1696}$) in Sect. 4.4 or the EHNP strategy (4.4) plus refinement (4.5) (giving $E = 2^{1622}$).

This is an instance of HNP-SUM, because we have samples $a_{i,j}$, small unknown multipliers $\delta_{i,j}$, and small errors $e_{i,j}$. We use the lattice approach detailed in Sect. 3 to recover the values of $\delta_{i,j}$ up to sign and division by a common and small factor. Because the $\delta_{i,j}$ involve bytes of cryptographic material and are essentially random, the greatest common divisor of the unknown multipliers in our attack is likely to be 1 or some very small value. It is therefore possible to brute force the sign and possible common factors to recover the $\delta_{i,j}$ exactly.

By examining the heuristic condition $T^{(n+1)/(n-1)}E \lesssim N$, we observe that $n = 3$ samples are necessary for the brute-force strategy, and $n = 2$ samples are necessary for the strategy of EHNP plus refinement.

4.7 Recovering Plaintexts

By combining the capabilities of Sects. 4.3 through 4.6, the adversary can learn $\delta_{i,j} = pt_i - pt_j$ for any pair (i,j) of plaintext blocks (up to sign). Note that recovering any single plaintext pt_i therefore reveals any other plaintext $pt_j = pt_i - \delta_{i,j}$. To accomplish this, we make use of the fact that $\ell(q)$ is 2 bytes of known plaintext and a property of the RSA equations.

When the public modulus e is small, it is easy to compute the most significant bits of the private modulus d. The least significant bits of d are not easy to compute, so this does not impact the security of RSA. To see why this is the case, observe that the RSA equation implies

$$\begin{aligned} d &\equiv e^{-1} \pmod{(p-1)(q-1)} \\ \Rightarrow ed - 1 &\equiv 0 \pmod{(p-1)(q-1)} \\ \Rightarrow ed - 1 &= k(p-1)(q-1) \\ \Rightarrow k &= e\frac{d}{(p-1)(q-1)} - \frac{1}{(p-1)(q-1)} \\ \Rightarrow k &\leq e. \end{aligned}$$

Thus if e is small, all possible values of k can be brute forced. A typical choice of e is 65537, which leads to an easy brute-force attack. MEGA's web client uses $e = 257$, and the SDK uses $e = 17$, so brute forcing k is even easier in this scenario. If k is known, then

$$\begin{aligned} d &= (k(p-1)(q-1)+1)/e \\ &= (k(pq-(p+q)+1)+1)/e \\ &= \frac{kN+k+1}{e} - \frac{p+q}{e}. \end{aligned}$$

The second term is unknown, but it is about as small as p and q, which are about half the size of d. The first term is known and with high probability reveals the most significant bits of d.

To use this in the attack, we first recover $\delta_{18,1} = pt_{18} - pt_1$. pt_{18} contains 16 significant bytes of d and pt_1 contains the length encoding $\ell(q)$. We guess all possible values of k from 1 to e, and for each guess, we determine what the significant bytes of d would be if that guess of k were correct. This gives a candidate value for pt_{18}, which we can use to compute a candidate pt_1. If the candidate pt_1 has valid length padding, the candidate pt_{18} may be correct. The odds of a false positive are acceptably small, around $e/2^{16}$, so for small e this is likely to reveal the true value of pt_{18}. Once pt_{18} is known, this reveals pt_j for every known $\delta_{18,j}$.

4.8 Recovering the Factorization

Section 4.7 demonstrates how to recover arbitrary plaintext blocks in the encoded RSA private key. This could be used to recover every plaintext block in the encoded key, but as in the attack of Backendal, Haller, and Paterson there is a more efficient solution to learning the factorization. We can recover every plaintext block corresponding to the most significant bytes of prime factor q, then use Coppersmith's method [7] to recover the full factorization.

For the 2048-bit modulus N with 1024-bit prime factors p and q, this requires at least 512 of the most significant bits. However, there is a trade-off between how many of the most significant bits are known, how complex the implementation is, and how long it takes the implementation to run. The proof-of-concept code for the original attack requires 683 bits and involves a dimension-3 lattice constructed with Coppersmith degree 2 and multiplicity 1. We improve the implementation by increasing the Coppersmith degree to 4 and multiplicity to 2, resulting in a lattice of dimension 5. Our improved implementation recovers the factorization with only 624 most significant bits. This corresponds to the most significant bits of q encoded in the first 5 plaintext blocks $pt_1, pt_2, \ldots, pt_5$ of the encoded private key. With the improved implementation, recovering these 5 plaintext values suffices to recover the full factorization.

4.9 Complexity

In this section, we analyze the overall complexity of both the *fast* attack requiring an expected 17 login attempts and the *small* attack requiring an expected 6.1 login attempts. Because both of our attacks share many steps, we begin by describing the overlap.

Both approaches assume that the 43 bytes returned by the client are at a fixed location in the output of the RSA decryption function, but this is optimistic. As described in Sect. 4.1, the client returns bytes 2-44 when byte 2 is nonzero, and bytes 3-45 otherwise. This can be modeled as the attacker querying an oracle which has some small probability of returning an incorrect answer. For both of our approaches, we assume that all s responses from the oracle are correct. Empirically, the analysis steps succeed when this is true and fails otherwise. If the analysis fails, the RSA ciphertext is re-randomized and the entire attack is repeated, collecting s fresh oracle responses. Under the simplifying assumption that the probability the oracle returns a correct response for a particular input is independently distributed and equal to 255/256 (byte 2 is nonzero), the probability that all s responses are correct is $(255/256)^s$. Therefore the expected number of oracle queries before the full attack is successful is $s(256/255)^s$.

Both approaches also overlap in the final stages of the attack, so much of the complexity analysis is repeated. For the Coppersmith attack in Sect. 4.8 to succeed, we assume the attack has successfully recovered 5 plaintext blocks $pt_1, \ldots, pt_5$. To acquire these 5 plaintexts, Sect. 4.7 processes differences between these plaintexts and a plaintext pt_{18} involving MSBs of RSA private exponent d. That is, this part of the attack requires knowledge of $\delta_{18,1}, \ldots, \delta_{18,5}$. These 5 values are obtained using the technique of Sect. 4.6 from five high-quality approximations.

The two approaches differ in how they obtain these five approximations.

Fast Attack. In the fast attack, we obtain the five high-quality approximations using Sect. 4.5 to refine 15 lower-quality approximations. For each high-quality approximation involving $\delta_{18,j}$, we assume we have lower-quality approximations of $\delta_{18,j}x$, $2^{128}\delta_{18,j}x$, and $2^{256}\delta_{18,j}x$ for a fixed and unknown x.

We obtain these lower-quality approximations using the EHNP technique in Sect. 4.4. This approach requires minimal guesswork, and it would still work if the 43 contiguous bytes were present at a different fixed offset. The disadvantage is that the EHNP transformation increases the error bounds, so we need more samples. As input to the EHNP transformation, we require 15 algebraic relationships involving $2^{128t}\delta_{18,j}$ for $t \in \{0, 1, 2\}$ and $j \in \{1, 2, \ldots, 5\}$.

As described in Sect. 4.3, each algebraic relationship involves taking the difference between two client responses involving different manipulations of the wrapped RSA private key. This naively means that the attack could be performed with 30 client interactions, but because each $\delta_{18,j}$ involves the same plaintext block pt_{18}, one single client response can be reused in all 15 client response pairs. In particular, the shared ciphertext leads to the client decoding the u value as

$$\mathrm{Decode}[\mathrm{D_{AES}}(ct_{33}) \mid \mathrm{D_{AES}}(ct_{18}) \mid \cdots \mid \mathrm{D_{AES}}(ct_{18}) \mid \mathrm{D_{AES}}(ct_{18}) \mid \mathrm{D_{AES}}(ct_{41})].$$

This results in a total of $s = 16$ error-free oracle responses sufficing to recover the RSA private key, or $16(256/255)^{16} \approx 17.03$ login attempts on average. None of the steps in this approach are particularly expensive, so the overall private key recovery is fast.

Small Attack. In the small attack, the five high-quality approximations are obtained by using the brute-force technique described in Sect. 4.4. The inputs to the brute-force technique are five algebraic relationships from Sect. 4.3, and brute force attempts to recover the unknown term $e'_1 - e''_1$, which can take on one of 511 values. Instead of trying all of the $(511)^5 \approx 2^{45}$ possibilities, we improve the complexity by focusing on three algebraic relationships at a time. This gives a more tractable brute-force cost of around 2^{27}.

For every combination of prefixes for the three algebraic relationships, we apply the lattice methods in Sect. 3 for $n = 3$ to recover candidate unknown multipliers. If this attempt succeeds and yields valid multipliers, the guessed prefixes may be correct. If the attempt fails, the guessed prefixes are probably incorrect. In practice, this approach reliably returns the correct prefixes.

Table 1. Average number of logins and average wall time required for each attack. The reported ranges represent a 95% confidence interval for the measured value.

Approach	Sample Size	Exp. Logins	Avg. Logins	Avg. Time (s)
Original [2]	10000	683	683 ± 0	9.46 ± 0.02
Fast Attack	10000	17.03	17.06 ± 0.08	5.59 ± 0.66
Small Attack	100	6.14	6.18 ± 0.20	16214 ± 522

Next, we take two samples with recovered prefixes and one sample with an unknown prefix and repeat the brute-force process to recover the unknown prefix. This is faster than brute forcing prefixes for three samples simultaneously. We repeat this process to recover all unknown prefixes. This results in five high-quality approximations from give algebraic relations.

Using the same argument as in the fast attack, the 5 algebraic relationships can be obtained using 6 correct oracle responses, which happens with probability $(255/256)^6 \approx 98\%$. The expected number of oracle responses needed for a successful attack would be $6(256/255)^6 \approx 6.14$. The most expensive step is brute forcing the triple of unknown prefixes, but this step is easily parallelized.

4.10 Experimental Evaluation

We benchmarked both of our new attacks[3] against the abstract proof-of-concept code of the attack in [2]. Both attacks are implemented in Python and use the lattice reduction implementation in SageMath. We ran all our attacks on an 88-core Intel Xeon E5-2699A processor running at 2.4 GHz. The original attack and our fast attack are single-threaded, and our small attack implementation is multithreaded. Table 1 reports a 95% confidence interval for each measurement.

As expected, there is good agreement between the measurements and the expected complexity calculated in Sect. 4.9. The measured time includes the time to simulate the client-server interactions, explaining why the original attack, which includes more login attempts but fewer analysis steps, takes longer on average to perform. The small attack takes an average of 4 h 30 min of wall-clock time to complete the analysis parallelized across 88 cores. Although this computational effort is not small, it is eminently tractable. We therefore conclude that the risk of these vulnerabilities was not limited to users who attempted to log in over 500 times, and instead show that users who attempted to log in at least 6 times may potentially be at risk. This illustrates the importance of updating clients to the latest patched version.

Table 2. Comparison of non-Coppersmith methods solving the implicit factoring problem. Our approach achieves the same heuristic bounds as in prior work with smaller lattices. Neither our bounds nor the bounds reported in prior work include higher-order terms to account for the approximation factor of lattice reduction algorithms, but experimentally the bounds are accurate for small to medium values of k.

Bits Shared	Approach	Bound	Rank	Dimension	Our Bound	Rank	Dimension
LSBs	[21]	$t \geq \frac{k}{k-1}\alpha$	k	k	$t \geq \frac{k}{k-1}\alpha$	$k+1$	$k+1$
MSBs	[9]	$t \geq \frac{k}{k-1}\alpha$	k	$\frac{k(k+1)}{2}$	$t \geq \frac{k}{k-1}\alpha$	$k+1$	$k+1$
Middle	[9]	$t \geq 2\frac{k}{k-1}\alpha$	$\frac{k(k+1)}{2}$	$\frac{k(k+1)}{2}$	$t \geq 2\frac{k}{k-1}\alpha$	$k+1$	$k+1$
LSBs and MSBs	-	-	-	-	$t \geq \frac{k}{k-1}\alpha$	$k+1$	$k+1$

5 Application: Implicit Factoring

In the implicit factoring problem, introduced by May and Ritzenhofen in 2009 [21], one wishes to factor k RSA moduli of the form $N_i = p_i q_i$ where the factors p_i share t bits in common, but the value of these bits is not known.

[3] Our implementation is available at https://github.com/keeganryan/attacks-poc.

This problem is typically considered in the context of unbalanced b-bit RSA moduli where $p_i \gg q_i$; the size of q_i is α bits

The original presentation considered the case of p_i sharing least significant bits (LSBs), and Sarkar and Maitra [27] expanded the definition to consider shared most significant bits (MSBs), a mix of shared LSBs and MSBs, and shared bits in the middle. They also gave Coppersmith-like approaches to solve these cases. Faugère, Marinier, and Renault [9] gave a simpler lattice construction for shared MSBs and shared middle bits, but they observed that their approach cannot be applied to moduli that have factors sharing a mix of LSBs and MSBs.

Apart from [21] and [9], methods to solve the implicit factoring problem have relied on Coppersmith-like techniques [20,25,27–29]. While these methods often yield superior bounds, they often require lattices of higher dimension, more involved analyses, and Gröbner basis calculations to recover the solutions to multivariate polynomial systems.

We show that HNP-SUM can be used to solve the implicit factoring problem when LSBs, MSBs, a mix of LSBs and MSBs, or middle bits are shared. While our lattice construction does not improve on the bounds of the Coppersmith-like techniques, it is the first non-Coppersmith technique to solve the mixed LSBs/MSBs problem, and it is the most efficient method to our knowledge which solves the shared middle bits problem for $k > 2$. Compared to the lattices of dimension $O(k^2)$ in [9], our lattice has rank and dimension $k+1$. All of our attacks achieve the same heuristic bounds as their non-Coppersmith counterparts, and a comparison of these approaches is given in Table 2.

5.1 LSBs or MSBs Shared

We begin by considering the case where LSBs are shared, MSBs are shared, or some mix are shared. The input to the problem is k RSA moduli $N_i = p_i q_i$ where $N_i < 2^b$, $q_i < \alpha$, and the p_i share several bits. Let the $t_1 \geq 0$ least significant and $t_2 \geq 0$ most significant bits be shared. This setup includes the cases where only LSBs or only MSBs are shared by setting t_1 or t_2 to 0. We have $p_i = p_{shared} + 2^{t_1}\tilde{p}_i$ where $\tilde{p}_i < 2^{b-\alpha-t_1-t_2}$. We rewrite this as

$$\begin{aligned} 2^{-t_1}N_i &\equiv 2^{-t_1}p_i q_i \\ &\equiv q_i(2^{-t_1}p_{shared}) + (\tilde{p}_i q_i) \qquad\qquad \pmod M \end{aligned}$$

where $M = 2^{b+t_1+t_2-\alpha} + 1$. Observe that this is an instance of HNP-SUM with samples $a_i = 2^{-t_1}N_i \bmod M$, unknown multipliers q_i, hidden number $x = 2^{-t_1}p_{shared} \bmod M$, and error $e_i = \tilde{p}_i q_i$. This gives bounds $T = 2^\alpha$ and $E = 2^{b-\alpha-t_1-t_2+\alpha}$, so Theorem 1 heuristically recovers the factors q_i when

$$(2^\alpha)^{(k+1)/(k-1)} 2^{b-t_1-t_2} \lesssim 2^{b+t_1+t_2-\alpha} + 1,$$

or equivalently

$$t_1 + t_2 \gtrsim \frac{k}{k-1}\alpha.$$

This gives a unified heuristic bound for the cases where LSBs are shared, MSBs are shared, or a mix of LSBs and MSBs are shared.

Justifying the Bounds. Although the choice of modulus M seems arbitrary, there is good justification for it. Note that the congruence would hold for larger choices of M, and a larger modulus would suggest the ability to solve HNP-SUM for larger T and E, and this would therefore imply the ability to solve the implicit factoring problem when arbitrarily few bits are shared. Increasing the modulus does improve the bounds up to a certain point, but this argument fails because beyond that point, Heuristic 2 is no longer satisfied. In particular, the projected sublattice of rank 2 contains a short vector of length $\approx 2^{b-\alpha}$ in violation of the Gaussian Heuristic. Since the sublattice recovery depends on the shortest vector in the projected sublattice, the ability to recover the sublattice is unchanged.

We experimentally observe that the point at which Heuristic 2 begins to fail is usually $\approx 2^{b+t_1+t_2-\alpha}$, and using a significantly larger modulus does not improve upon the predicted bounds. In practice, we set M to be slightly larger because there is a small chance that Heuristic 2 holds for a slightly larger modulus, and making M larger by a handful of bits barely affects running time. We also make M odd to ensure 2^{-t_1} exists in the ring of integers modulo M.

5.2 Middle Bits Shared

We next consider the case where we are given k RSA moduli $N_i = p_i q_i$ with $N_i < 2^b$, $q_i < 2^\alpha$ and the $(l, l+t)$ middle bits of p_i are shared. That is, $p_i = \tilde{p}'_i 2^{l+t} + p_{mid} 2^l + \tilde{p}_i$ where $\tilde{p}'_i < 2^{b-\alpha-l-t}$ and $\tilde{p}_i < 2^l$. We rewrite this as

$$N_i \equiv 2^{l+t}\tilde{p}'_i q_i + 2^l p_{mid} q_i + \tilde{p}_i q_i \equiv q_i(2^l p_{mid}) + (\tilde{p}_i q_i) \pmod{2^{l+t}}$$

Table 3. Shared Least Significant Bits. We compared our lattice construction for solving implicit factoring against [21] for $b = 2048$, $\alpha = 512$, and various shared bits t. The row in bold represents the first value of t for which the condition $t \geq \frac{k}{k-1}\alpha$ is satisfied and we expect the lattice methods to succeed. We see that our approach is approximately as powerful as [21] or a bit stronger, and the success rate follows the predicted bound to within a couple of bits.

$k = 2$			$k = 5$			$k = 30$		
Leakage (t)	[21]	Ours	Leakage (t)	[21]	Ours	Leakage (t)	[21]	Ours
1022	0%	0%	638	0%	0%	528	0%	0%
1023	0%	0%	639	0%	0%	529	0%	0%
1024	**35%**	**44%**	**640**	**0%**	**0%**	**530**	**0%**	**0%**
1025	100%	100%	641	7%	34%	531	0%	78%
1026	100%	100%	642	89%	96%	532	53%	100%
1027	100%	100%	643	100%	100%	533	100%	100%
1028	100%	100%	644	100%	100%	534	100%	100%

Table 4. Shared Most Significant Bits. We compare our construction against [9] for $b = 2048$, $\alpha = 512$, and various t. The row in bold represents the first value of t for which $t \geq \frac{k}{k-1}\alpha$. As was the case for LSBs, our performance is close to [9] and the predicted bound, although this time it is slightly weaker.

$k = 2$			$k = 5$			$k = 30$		
Leakage (t)	[9]	Ours	Leakage (t)	[9]	Ours	Leakage (t)	[9]	Ours
1022	0%	0%	638	0%	0%	528	0%	0%
1023	0%	0%	639	0%	0%	529	0%	0%
1024	**9%**	**2%**	**640**	**0%**	**0%**	**530**	**0%**	**0%**
1025	71%	28%	641	5%	1%	531	50%	34%
1026	100%	98%	642	75%	43%	532	98%	97%
1027	100%	100%	643	99%	96%	533	100%	100%
1028	100%	100%	644	100%	100%	534	100%	100%

and observe that this gives an instance of HNP-SUM with $a_i = N_i$, $t_i = q_i$, $x = (2^l p_{mid})$, and $e_i = \tilde{p}_i q_i$. This gives bounds $T = 2^\alpha$ and $E = 2^{\alpha+l}$, so Theorem 1 heuristically recovers the t_i when

$$(2^\alpha)^{(k+1)/(k-1)} 2^{\alpha+l} \lesssim 2^{l+t} \Leftrightarrow t \gtrsim \frac{2k}{k-1}\alpha.$$

5.3 Experimental Evaluation

We implemented our reductions from implicit factoring to HNP-SUM and the lattice methods described in [21] and [9]. We performed experiments on 2048-bit moduli for $k \in \{2, 5, 30\}$ and several t around the boundary for which we predict the instance is solvable. In all cases, we find that our predicted bound is within a couple bits of what we observed. We attempted to solve 100 instances for each combination of parameters and report the results in Tables 3 through 6.

Our implementation was mostly written in Python and Sage. We also use a custom C++ lattice reduction implementation. We ran each attack instance on a single thread of an 88-core Intel Xeon E5-2699A processor running at 2.4 GHz. While our attack averaged under a second in all cases and the prior approaches were similarly fast in most cases, [9] was significantly slower for $k = 30$.

This was primarily due to the cost of lattice reduction. When solving the case of shared MSBs with $k = 30$, we reduce a lattice of rank 30, dimension 465, and entries of size 2048 bits. In the case of shared middle bits, both the rank and dimension are 465. Our custom lattice reduction implementation took around 10 s per instance in the first case and 4 min in the second.

Our experiments demonstrate that our heuristically derived bounds are accurate for a variety of parameters. Our methods are more efficient than prior work, and our reduction to HNP-SUM provides a straightforward lattice-based cryptanalysis to solve the implicit factoring problem in all shared-bit contexts.

Table 5. Shared MSBs and LSBs. We determine the success rate of our construction for $b = 2048$, $\alpha = 512$, and various t with the shared bits split evenly between the MSBs and LSBs. There is no non-Coppersmith method we are aware of to compare against, but the performance of our method closely approximates the predicted bound.

$k = 2$		$k = 5$		$k = 30$	
Leakage (t)	Ours	Leakage (t)	Ours	Leakage (t)	Ours
1022	0%	638	0%	528	0%
1023	0%	639	0%	529	0%
1024	**2%**	**640**	**0%**	**530**	**0%**
1025	36%	641	0%	531	36%
1026	91%	642	51%	532	95%
1027	100%	643	98%	533	100%
1028	100%	644	100%	534	100%

Table 6. Shared Middle Bits. We compare our construction against [9] for $b = 2048$, $\alpha = 380$, and various t around the boundary $t \geq 2\frac{k}{k-1}\alpha$. We find that our approach closely matches the predicted bound. However, the approach of [9] for $k = 30$ fails for all these values of t. This is because the lattice approximation factor is quite significant for lattices of rank $k(k+1)/2 = 465$, and lattice reduction failed to find the shortest vector for these parameters.

$k = 2$			$k = 5$			$k = 30$		
Leakage (t)	[9]	Ours	Leakage (t)	[9]	Ours	Leakage (t)	[9]	Ours
1518	0%	0%	948	0%	0%	785	0%	0%
1519	1%	1%	949	0%	0%	786	0%	0%
1520	**7%**	**10%**	**950**	**0%**	**0%**	**787**	**0%**	**0%**
1521	35%	38%	951	0%	0%	788	0%	0%
1522	69%	66%	952	9%	6%	789	0%	11%
1523	88%	90%	953	40%	38%	790	0%	44%
1524	91%	93%	954	68%	67%	791	0%	53%
1525	96%	96%	955	86%	84%	792	0%	67%
1526	100%	99%	956	93%	93%	793	0%	89%
1527	99%	99%	957	93%	93%	794	0%	93%

Acknowledgment. We thank Miro Haller and Kenny Paterson for their helpful comments on an earlier draft, insightful discussions, and providing further context. This material is based upon work supported by the National Science Foundation under grants no. 2048563 and 1913210.

References

1. Alexi, W., Chor, B., Goldreich, O., Schnorr, C.P.: RSA and Rabin functions: certain parts are as hard as the whole. SIAM J. Comput. **17**(2), 194–209 (1988). https://doi.org/10.1137/0217013
2. Backendal, M., Haller, M., Paterson, K.G.: MEGA: malleable encryption goes awry. In: 2023 IEEE Symposium on Security and Privacy (SP), pp. 450–467 (2023). https://doi.org/10.1109/SP46215.2023.00026
3. Bauer, A., Joux, A.: Toward a rigorous variation of coppersmith's algorithm on three variables. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 361–378. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72540-4_21
4. Boneh, D., Halevi, S., Howgrave-Graham, N.: The modular inversion hidden number problem. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 36–51. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_3
5. Boneh, D., Shparlinski, I.E.: On the unpredictability of bits of the elliptic curve Diffie-Hellman scheme. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 201–212. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_12
6. Boneh, D., Venkatesan, R.: Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 129–142. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_11
7. Coppersmith, D.: Finding a small root of a bivariate integer equation; factoring with high bits known. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 178–189. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68339-9_16
8. De Mulder, E., Hutter, M., Marson, M.E., Pearson, P.: Using Bleichenbacher's solution to the hidden number problem to attack nonce leaks in 384-bit ECDSA. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 435–452. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40349-1_25
9. Faugère, J.-C., Marinier, R., Renault, G.: Implicit factoring with shared most significant and middle bits. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 70–87. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13013-7_5
10. Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 31–51. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_3
11. Garner, H.L.: The residue number system. In: Papers Presented at the the 3–5 March 1959, Western Joint Computer Conference, pp. 146–153. IRE-AIEE-ACM 1959 (Western), Association for Computing Machinery, New York, NY, USA (1959). https://doi.org/10.1145/1457838.1457864
12. Hlaváč, M., Rosa, T.: Extended hidden number problem and its cryptanalytic applications. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 114–133. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74462-7_9
13. Howgrave-Graham, N.A., Smart, N.P.: Lattice attacks on digital signature schemes. Des. Codes Crypt. **23**(3), 283–290 (2001). https://doi.org/10.1023/A:1011214926272
14. Howgrave-Graham, N.: Finding small roots of univariate modular equations revisited. In: Darnell, M. (ed.) Cryptography and Coding 1997. LNCS, vol. 1355, pp. 131–142. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0024458

15. Howgrave-Graham, N.: Approximate integer common divisors. In: Silverman, J.H. (ed.) CaLC 2001. LNCS, vol. 2146, pp. 51–66. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44670-2_6
16. Howgrave-Graham, N.A., Nguyen, P.Q., Shparlinski, I.E.: Hidden number problem with hidden multipliers, timed-release crypto, and noisy exponentiation. Math. Comput. **72**(243), 1473–1485 (2003)
17. Jutla, C.S.: On finding small solutions of modular multivariate polynomial equations. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 158–170. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054124
18. Kannan, R., Bachem, A.: Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. SIAM J. Comput. **8**(4), 499–507 (1979). https://doi.org/10.1137/0208040
19. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Math. Ann. **261**(4), 515–534 (1982). https://doi.org/10.1007/BF01457454
20. Lu, Y., Peng, L., Zhang, R., Hu, L., Lin, D.: Towards optimal bounds for implicit factorization problem. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. LNCS, vol. 9566, pp. 462–476. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31301-6_26
21. May, A., Ritzenhofen, M.: Implicit factoring: on polynomial time factoring given only an implicit hint. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 1–14. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00468-1_1
22. Nguyen, P.Q.: Hermite's constant and lattice algorithms. In: Nguyen, P., (eds.) The LLL Algorithm. Information Security and Cryptography, pp. 19–69. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-02295-1
23. Nguyen, P.Q., Stehlé, D.: LLL On the average. In: Hess, F., Pauli, S., Pohst, M. (eds.) ANTS 2006. LNCS, vol. 4076, pp. 238–256. Springer, Heidelberg (2006). https://doi.org/10.1007/11792086_18
24. Ortmann, M.: MEGA security update (2022). https://blog.mega.io/mega-security-update/
25. Peng, L., Hu, L., Xu, J., Huang, Z., Xie, Y.: Further improvement of factoring RSA moduli with implicit hint. In: Pointcheval, D., Vergnaud, D. (eds.) AFRICACRYPT 2014. LNCS, vol. 8469, pp. 165–177. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06734-6_11
26. Ryan, K., Heninger, N.: The hidden number problem with small unknown multipliers: cryptanalyzing MEGA in six queries and other applications. Cryptology ePrint Archive, Report 2022/914 (2022). https://eprint.iacr.org/2022/914
27. Sarkar, S., Maitra, S.: Further results on implicit factoring in polynomial time. Adv. Math. Commun. **3**(2), 205–217 (2009). https://doi.org/10.3934/amc.2009.3.205
28. Sarkar, S., Maitra, S.: Approximate integer common divisor problem relates to implicit factorization. IEEE Trans. Inf. Theory **57**(6), 4002–4013 (2011). https://doi.org/10.1109/TIT.2011.2137270
29. Wang, S., Qu, L., Li, C., Fu, S.: A better bound for implicit factorization problem with shared middle bits. Sci. Chin. Inf. Sci. **61**(3), 1–10 (2017). https://doi.org/10.1007/s11432-017-9176-5