Finding Small Solutions to Low Degree Polynomials and Applications

Aleksei Udovenko

SnT, University of Luxembourg

Seminar on Lattices in Cryptography

June 7, 2019

UNIVERSITÉ DU LUXEMBOURG

securityandtrust.lu

Plan

Finding Small Solutions

Applications to RSA

Conclusion

Goal

Theorem

Let N be an integer and $f \in \mathbb{Z}_N[x]$ monic, $\deg f = d$*.*
Then we can efficiently find all

$$x \in \mathbb{Z} : |x| \leq B \text{ and } f(x) \equiv 0 \pmod{N}$$

for $B = N^{1/d}$.

Main Idea

Finding roots over $\mathbb{Z}_N$?

Main Idea

Finding roots over $\mathbb{Z}_N$?

Finding roots over $\mathbb{Z}$ is easy.

Main Idea

Finding roots over $\mathbb{Z}_N$?

Finding roots over $\mathbb{Z}$ is easy.

Let's find $g \in \mathbb{Z}[x]$, such that

$$f(x) \equiv 0 \pmod{N} \Rightarrow g(x) = 0.$$

Main Idea

How? We want to prevent overflowing N:

$$\text{Let } x \in \mathbb{Z}, |x| < B.$$
$$If |g(x)| < N, \text{ then}$$
$$g(x) \equiv 0 \pmod{N} \Rightarrow g(x) = 0.$$

Main Idea

How? We want to prevent overflowing N:

$$\text{Let } x \in \mathbb{Z}, |x| < B.$$
$$\textit{If} \, |g(x)| < N, \text{ then}$$
$$g(x) \equiv 0 \pmod{N} \Rightarrow g(x) = 0.$$

Let

$$g(x) = x^d + a_{d-1}x^{d-1} + \ldots + a_1 x + a_0 \in \mathbb{Z}[x].$$

Main Idea

How? We want to prevent overflowing N:

$$\text{Let } x \in \mathbb{Z}, |x| < B.$$
$$If |g(x)| < N, \text{ then}$$
$$g(x) \equiv 0 \pmod{N} \Rightarrow g(x) = 0.$$

Let

$$g(x) = x^d + a_{d-1}x^{d-1} + \ldots + a_1 x + a_0 \in \mathbb{Z}[x].$$

We want for all i and for all x with $|x| \leq B$:

$$|a_i x^i| < \frac{N}{d+1} \quad \Leftarrow \quad |a_i| < \frac{1}{B^i}\frac{N}{d+1}.$$

Main Idea

How? We want to prevent overflowing N:

$$\text{Let } x \in \mathbb{Z}, |x| < B.$$
$$\textit{If} \, |g(x)| < N, \text{ then}$$
$$g(x) \equiv 0 \pmod{N} \Rightarrow g(x) = 0.$$

Let

$$g(x) = x^d + a_{d-1}x^{d-1} + \ldots + a_1 x + a_0 \in \mathbb{Z}[x].$$

We want for all i and for all x with $|x| \leq B$:

$$|a_i x^i| < \frac{N}{d+1} \quad \Leftarrow \quad \boxed{|a_i| < \frac{1}{B^i}\frac{N}{d+1}.}$$

Idea 1: Constant Multiples

Consider a (nonzero) multiple of $f(x)$. It has same roots modulo N.

Johan Håstad. 1988. Solving simultaneous modular equations of low degree.
SIAM J. Comput. 17, 2 (April 1988), 336-341.

Idea 1: Constant Multiples

Consider a (nonzero) multiple of $f(x)$. It has same roots modulo N.

How to find a "good" multiple? Use LLL!

Johan Håstad. 1988. Solving simultaneous modular equations of low degree.
SIAM J. Comput. 17, 2 (April 1988), 336-341.

Idea 1: Constant Multiples

Consider the lattice $\mathcal{L}$:

$$\begin{pmatrix} N & 0 & \cdots & 0 & a_0 \\ 0 & N & \ddots & 0 & a_1 \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & N & a_{d-1} \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix}_{(d+1)\times(d+1)}$$

Idea 1: Constant Multiples

Consider the lattice $\mathcal{L}$:

$$\begin{pmatrix} N & 0 & \cdots & 0 & a_0 \\ 0 & N & \ddots & 0 & a_1 \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & N & a_{d-1} \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix}_{(d+1)\times(d+1)}$$

The last column corresponds to $f(x)$.
The other columns correspond
to reductions $\mod N$ (into $\left[-\frac{N}{2}; \frac{N}{2}\right]$) .

Idea 1: Constant Multiples

Consider the lattice $\mathcal{L}$:

$$\begin{pmatrix} N & 0 & \cdots & 0 & a_0 \\ 0 & N & \ddots & 0 & a_1 \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & N & a_{d-1} \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix}_{(d+1)\times(d+1)}$$

a short vector $v \in \mathcal{L}$:

$$\begin{pmatrix} ka_0 \mod N \\ ka_1 \mod N \\ \vdots \\ ka_{d-1} \mod N \\ k \mod N \end{pmatrix}_{(d+1)x1}$$

The last column corresponds to $f(x)$.
The other columns correspond
to reductions $\mod N$ (into $\left[-\frac{N}{2}; \frac{N}{2}\right]$) .

Idea 1: Constant Multiples

Consider the lattice $\mathcal{L}$:

$$\begin{pmatrix} N & 0 & \cdots & 0 & a_0 \\ 0 & BN & \ddots & 0 & Ba_1 \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & B^{d-1}N & B^{d-1}a_{d-1} \\ 0 & \cdots & \cdots & 0 & B^d \end{pmatrix}$$

a short vector $v \in \mathcal{L}$:

$$\begin{pmatrix} ka_0 \mod N \\ (ka_1 \mod N)B \\ \vdots \\ (ka_{d-1} \mod N)B^{d-1} \\ (k \mod N)B^d \end{pmatrix}$$

Recall that we want $B^i a'_i < \frac{N}{d+1}$ (a'_i is a coefficient in new polynomial).

Idea 1: Constant Multiples

Consider the lattice $\mathcal{L}$:

$$\begin{pmatrix} N & 0 & \cdots & 0 & a_0 \\ 0 & BN & \ddots & 0 & Ba_1 \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & B^{d-1}N & B^{d-1}a_{d-1} \\ 0 & \cdots & \cdots & 0 & B^d \end{pmatrix}$$

a short vector $v \in \mathcal{L}$:

$$\begin{pmatrix} ka_0 \mod N \\ (ka_1 \mod N)B \\ \vdots \\ (ka_{d-1} \mod N)B^{d-1} \\ (k \mod N)B^d \end{pmatrix}$$

Recall that we want $B^i a'_i < \frac{N}{d+1}$ (a'_i is a coefficient in new polynomial).

Scale coordinates! $\Rightarrow$ minimize $B^i a'_i = (ka_i \mod N)B^i$.

Idea 1: Constant Multiples

Consider the lattice $\mathcal{L}$:

$$\begin{pmatrix} N & 0 & \cdots & 0 & a_0 \\ 0 & BN & \ddots & 0 & Ba_1 \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & B^{d-1}N & B^{d-1}a_{d-1} \\ 0 & \cdots & \cdots & 0 & B^d \end{pmatrix}$$

a short vector $v \in \mathcal{L}$:

$$\begin{pmatrix} ka_0 \mod N \\ (ka_1 \mod N)B \\ \vdots \\ (ka_{d-1} \mod N)B^{d-1} \\ (k \mod N)B^d \end{pmatrix}$$

Upper-triangular structure:

$$\det(\mathcal{L}) = N \cdot BN \cdot \ldots \cdot B^{d-1}N \cdot B^d = N^d B^{d(d+1)/2}.$$

Idea 1: Constant Multiples

$$\det(\mathcal{L}) = N \cdot BN \cdot \ldots \cdot B^{d-1}N \cdot B^d = N^d B^{d(d+1)/2}.$$

Idea 1: Constant Multiples

$$\det(\mathcal{L}) = N \cdot BN \cdot \ldots \cdot B^{d-1}N \cdot B^d = N^d B^{d(d+1)/2}.$$

LLL: $v_1 \in \mathcal{L}$: $\quad \|v_1\| \leq 2^{\frac{d}{4}} \cdot \det(\mathcal{L})^{\frac{1}{d+1}} = 2^{\frac{d}{4}} \cdot N^{d/(d+1)} B^{d/2}.$

Idea 1: Constant Multiples

$$\det(\mathcal{L}) = N \cdot BN \cdot \ldots \cdot B^{d-1}N \cdot B^d = N^d B^{d(d+1)/2}.$$

$$\text{LLL: } v_1 \in \mathcal{L}: \quad \|v_1\| \leq 2^{\frac{d}{4}} \cdot \det(\mathcal{L})^{\frac{1}{d+1}} = 2^{\frac{d}{4}} \cdot N^{d/(d+1)} B^{d/2}.$$

Require $\|v_1\| < \frac{N}{d+1}$:

$$2^{\frac{d}{4}} \cdot N^{d/(d+1)} B^{d/2} < \frac{N}{d+1} \quad \Leftrightarrow \quad B < \left(\frac{N^{1/(d+1)}}{(d+1) \cdot 2^{d/4}} \right)^{\frac{2}{d}}$$

Idea 1: Constant Multiples

$$\det(\mathcal{L}) = N \cdot BN \cdot \ldots \cdot B^{d-1}N \cdot B^d = N^d B^{d(d+1)/2}.$$

$$\text{LLL: } v_1 \in \mathcal{L} : \qquad \|v_1\| \leq 2^{\frac{d}{4}} \cdot \det(\mathcal{L})^{\frac{1}{d+1}} = 2^{\frac{d}{4}} \cdot N^{d/(d+1)} B^{d/2}.$$

Require $\|v_1\| < \frac{N}{d+1}$:

$$2^{\frac{d}{4}} \cdot N^{d/(d+1)} B^{d/2} < \frac{N}{d+1} \quad \Leftrightarrow \quad B < \left(\frac{N^{1/(d+1)}}{(d+1) \cdot 2^{d/4}} \right)^{\frac{2}{d}}$$

$$\Leftrightarrow \quad B < \frac{N^{\frac{2}{d(d+1)}}}{\sqrt{2}(d+1)^{2/d}} = \mathcal{O}(N^{\frac{2}{d(d+1)}}).$$

Idea 2: Variable/Polynomial Multiples

$$f(x) = 0 \mod N \quad \Rightarrow \quad h(x)f(x) = 0 \mod N,$$
$$\text{for any } h \in \mathbb{Z}_N[x].$$

D. Coppersmith. 1997. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. Journal of Cryptology, 10:233260, 1997.

Idea 2: Variable/Polynomial Multiples

$$f(x) = 0 \mod N \quad \Rightarrow \quad h(x)f(x) = 0 \mod N,$$
$$\text{for any } h \in \mathbb{Z}_N[x].$$

$$\text{basis: } x^i f(x) \text{ for } i \in \{0, \ldots, t\}.$$

D. Coppersmith. 1997. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. Journal of Cryptology, 10:233260, 1997.

Idea 2: Variable/Polynomial Multiples

$$f(x) = 0 \mod N \quad \Rightarrow \quad h(x)f(x) = 0 \mod N,$$
$$\text{for any } h \in \mathbb{Z}_N[x].$$

basis: $x^i f(x)$ for $i \in \{0, \ldots, t\}$.

Let's add $x^i f(x)$ to the lattice, for $i \in \{1, \ldots, d-1\}$.

D. Coppersmith. 1997. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. Journal of Cryptology, 10:233260, 1997.

Idea 2: Variable/Polynomial Multiples

$$f(x) = 0 \mod N \quad \Rightarrow \quad h(x)f(x) = 0 \mod N,$$
$$\text{for any } h \in \mathbb{Z}_N[x].$$

basis: $x^i f(x)$ for $i \in \{0, \ldots, t\}$.

Let's add $x^i f(x)$ to the lattice, for $i \in \{1, \ldots, d-1\}$.

Careful: increases degree!

D. Coppersmith. 1997. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. Journal of Cryptology, 10:233260, 1997.

Idea 2: Variable/Polynomial Multiples

$$
\begin{array}{c}
 \\ 1 \\ x \\ x^2 \\ \vdots \\ x^{d-1} \\ x^d \\ \vdots \\ x^{2d-1}
\end{array}
\begin{array}{c}
\begin{array}{cccccccc}
x^0 & x^1 & \cdots & x^{d-1} & f(x) & x \cdot f(x) & \cdots & x^{d-1} \cdot f(x)
\end{array} \\
\hline
\begin{pmatrix}
N & 0 & 0 & 0 & a_0 & 0 & \dots & 0 \\
0 & BN & \ddots & \vdots & \vdots & Ba_0 & \ddots & \vdots \\
\vdots & 0 & \ddots & 0 & \vdots & \vdots & \ddots & 0 \\
\vdots & \vdots & \ddots & B^{d-1}N & B^{d-1}a_{d-1} & \vdots & \vdots & B^{d-1}a_0 \\
\vdots & \vdots & \ddots & 0 & B^d & B^d a_{d-1} & \vdots & \vdots \\
\vdots & \vdots & \ddots & \ddots & \ddots & B^{d+1} & \ddots & \vdots \\
\vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & B^{2d-2}a_{d-1} \\
0 & 0 & \cdots & \cdots & 0 & 0 & 0 & B^{2d-1}
\end{pmatrix}
\end{array}
$$

Idea 2: Variable/Polynomial Multiples

$$
\begin{array}{c|cccccccc}
 & x^0 & x^1 & \cdots & x^{d-1} & f(x) & x \cdot f(x) & \cdots & x^{d-1} \cdot f(x) \\
\hline
1 & N & 0 & 0 & 0 & a_0 & 0 & \cdots & 0 \\
x & 0 & BN & \ddots & \vdots & \vdots & Ba_0 & \ddots & \vdots \\
x^2 & \vdots & 0 & \ddots & 0 & \vdots & \vdots & \ddots & 0 \\
\vdots & \vdots & \vdots & \ddots & B^{d-1}N & B^{d-1}a_{d-1} & \vdots & \vdots & B^{d-1}a_0 \\
x^{d-1} & \vdots & \vdots & \ddots & 0 & B^d & B^d a_{d-1} & \vdots & \vdots \\
x^d & \vdots & \vdots & \ddots & \ddots & \ddots & B^{d+1} & \ddots & \vdots \\
\vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & B^{2d-2}a_{d-1} \\
x^{2d-1} & 0 & 0 & \cdots & \cdots & 0 & 0 & 0 & B^{2d-1}
\end{array}
$$

$$\det(\mathcal{L}) = N \cdot BN \cdot \ldots \cdot B^{d-1}N \cdot B^d \cdot B^{d+1} \cdot \ldots \cdot B^{2d-1} = N^d B^{d(2d-1)}.$$

Idea 2: Variable/Polynomial Multiples

$$\det(\mathcal{L}) = N^d B^{d(2d-1)}.$$

Idea 2: Variable/Polynomial Multiples

$$\det(\mathcal{L}) = N^d B^{d(2d-1)}.$$

LLL: $v_1 \in \mathcal{L}$: $\quad \|v_1\| \leq 2^{\frac{2d-1}{4}} \cdot \det(\mathcal{L})^{\frac{1}{2d}} = 2^{\frac{2d-1}{4}} \cdot N^{1/2} B^{(2d-1)/2}.$

Idea 2: Variable/Polynomial Multiples

$$\det(\mathcal{L}) = N^d B^{d(2d-1)}.$$

$$\text{LLL: } v_1 \in \mathcal{L} : \quad \|v_1\| \leq 2^{\frac{2d-1}{4}} \cdot \det(\mathcal{L})^{\frac{1}{2d}} = 2^{\frac{2d-1}{4}} \cdot N^{1/2} B^{(2d-1)/2}.$$

Require $\|v_1\| < \frac{N}{2d}$:

$$2^{\frac{2d-1}{4}} \cdot N^{1/2} B^{(2d-1)/2} < \frac{N}{2d} \quad \Leftrightarrow \quad B < \left(\frac{N^{1/2}}{2d \cdot 2^{(2d-1)/4}} \right)^{\frac{2}{2d-1}}$$

Idea 2: Variable/Polynomial Multiples

$$\det(\mathcal{L}) = N^d B^{d(2d-1)}.$$

$$\text{LLL: } v_1 \in \mathcal{L} : \quad \|v_1\| \le 2^{\frac{2d-1}{4}} \cdot \det(\mathcal{L})^{\frac{1}{2d}} = 2^{\frac{2d-1}{4}} \cdot N^{1/2} B^{(2d-1)/2}.$$

Require $\|v_1\| < \frac{N}{2d}$:

$$2^{\frac{2d-1}{4}} \cdot N^{1/2} B^{(2d-1)/2} < \frac{N}{2d} \quad \Leftrightarrow \quad B < \left(\frac{N^{1/2}}{2d \cdot 2^{(2d-1)/4}} \right)^{\frac{2}{2d-1}}$$

$$\Leftrightarrow \quad B < \frac{N^{\frac{1}{2d-1}}}{2\sqrt{2} \cdot d^{2/(2d-1)}} = \mathcal{O}(N^{\frac{1}{2d-1}}).$$

Idea 3: f-Multiples $\pmod{N^m}$

Idea 1: $\mathcal{L} \leftarrow \{f(x)\} \cup \left\{N, Nx, Nx^2, \ldots, Nx^{d-1}\right\}.$

Idea 2: $\mathcal{L} \leftarrow \ldots \cup \left\{f(x), xf(x), \ldots, x^{d-1}f(x)\right\}$

(increase degree of the polynomials).

Idea 3: $\mathcal{L} \leftarrow$???

Idea 3: f-Multiples $\pmod{N^m}$

Idea 1: $\mathcal{L} \leftarrow \{f(x)\} \cup \left\{N, Nx, Nx^2, \ldots, Nx^{d-1}\right\}.$

Idea 2: $\mathcal{L} \leftarrow \ldots \cup \left\{f(x), xf(x), \ldots, x^{d-1}f(x)\right\}$

(increase degree of the polynomials).

Idea 3: increase degree of N :

consider polynomials mod N^m.

Idea 3: f-Multiples $\pmod{N^m}$

Idea 1: $\mathcal{L} \leftarrow \{f(x)\} \cup \left\{N, Nx, Nx^2, \ldots, Nx^{d-1}\right\}$.

Idea 2: $\mathcal{L} \leftarrow \ldots \cup \left\{f(x), xf(x), \ldots, x^{d-1}f(x)\right\}$

(increase degree of the polynomials).

Idea 3: increase degree of N :

consider polynomials mod N^m.

powers of $f(x)$ allow to lift:

$\mathcal{L} \leftarrow \left\{N^{m-i}f(x)^i x^j \mid 0 \leq i \leq m, 0 \leq j \leq d-1\right\}$.

covers Ideas 1 and 2!

Idea 3: f-Multiples $\pmod{N^m}$

Example: $d = 2, m = 2$

$$\begin{array}{c} \\ x^0 \\ x^1 \\ x^2 \\ x^3 \\ x^4 \\ x^5 \end{array}
\begin{array}{c}
\begin{array}{cccccc} N^2x^0 & N^2x^1 & N^1x^0f(x)^1 & N^1x^1f(x)^1 & x^0f(x)^2 & x^1f(x)^2 \end{array} \\
\begin{pmatrix}
N^2 & ? & ? & ? & ? & ? \\
0 & N^2B^1 & ? & ? & ? & ? \\
0 & 0 & N^1B^2 & ? & ? & ? \\
0 & 0 & 0 & N^1B^3 & ? & ? \\
0 & 0 & 0 & 0 & B^4 & ? \\
0 & 0 & 0 & 0 & 0 & B^5
\end{pmatrix}
\end{array}$$

Idea 3: f-Multiples $\pmod{N^m}$

Example: $d = 2, m = 2$

$$
\begin{array}{c}
\\ x^0 \\ x^1 \\ x^2 \\ x^3 \\ x^4 \\ x^5
\end{array}
\begin{array}{cccccc}
N^2x^0 & N^2x^1 & N^1x^0f(x)^1 & N^1x^1f(x)^1 & x^0f(x)^2 & x^1f(x)^2 \\
\end{array}
$$

$$
\begin{array}{c} x^0 \\ x^1 \\ x^2 \\ x^3 \\ x^4 \\ x^5 \end{array}
\begin{pmatrix}
N^2 & ? & ? & ? & ? & ? \\
0 & N^2B^1 & ? & ? & ? & ? \\
0 & 0 & N^1B^2 & ? & ? & ? \\
0 & 0 & 0 & N^1B^3 & ? & ? \\
0 & 0 & 0 & 0 & B^4 & ? \\
0 & 0 & 0 & 0 & 0 & B^5
\end{pmatrix}
$$

Same upper-triangular structure $\Rightarrow$ easy calculation of $\det(\mathcal{L})$.

Idea 3: f-Multiples $\pmod{N^m}$

$$\dim(\mathcal{L}) = d(m+1),$$
$$\det(\mathcal{L}) = N^{dm(m+1)/2} B^{(d(m+1)-1)d(m+1)/2}.$$

Idea 3: f-Multiples $\pmod{N^m}$

$$\begin{aligned}\dim(\mathcal{L}) &= d(m+1),\\ \det(\mathcal{L}) &= N^{dm(m+1)/2} B^{(d(m+1)-1)d(m+1)/2}.\end{aligned}$$

$$\text{LLL: } \|v_1\| \leq 2^{\frac{d(m+1)-1}{4}} \cdot \det(\mathcal{L})^{\frac{1}{d(m+1)}} = 2^{\frac{d(m+1)-1}{4}} \cdot N^{m/2} B^{(d(m+1)-1)/2}.$$

Idea 3: f-Multiples $\pmod{N^m}$

$$\dim(\mathcal{L}) = d(m+1),$$
$$\det(\mathcal{L}) = N^{dm(m+1)/2} B^{(d(m+1)-1)d(m+1)/2}.$$

$$\text{LLL: } \|v_1\| \leq 2^{\frac{d(m+1)-1}{4}} \cdot \det(\mathcal{L})^{\frac{1}{d(m+1)}} = 2^{\frac{d(m+1)-1}{4}} \cdot N^{m/2} B^{(d(m+1)-1)/2}.$$

Require $\|v_1\| < \frac{N^m}{d(m+1)}$:

$$2^{\frac{d(m+1)-1}{4}} \cdot N^{m/2} B^{(d(m+1)-1)/2} < \frac{N^m}{d(m+1)} \quad \Leftrightarrow$$

Idea 3: f-Multiples $\pmod{N^m}$

$$\dim(\mathcal{L}) = d(m+1),$$
$$\det(\mathcal{L}) = N^{dm(m+1)/2} B^{(d(m+1)-1)d(m+1)/2}.$$

$$\text{LLL: } \|v_1\| \leq 2^{\frac{d(m+1)-1}{4}} \cdot \det(\mathcal{L})^{\frac{1}{d(m+1)}} = 2^{\frac{d(m+1)-1}{4}} \cdot N^{m/2} B^{(d(m+1)-1)/2}.$$

Require $\|v_1\| < \frac{N^m}{d(m+1)}$:

$$2^{\frac{d(m+1)-1}{4}} \cdot N^{m/2} B^{(d(m+1)-1)/2} < \frac{N^m}{d(m+1)} \quad \Leftrightarrow$$

$$\Leftrightarrow \quad B < \alpha(d,m) \cdot N^{\frac{m}{d(m+1)-1}} = \alpha(d,m) \cdot N^{\frac{1}{d} - \Theta(\frac{1}{m})}.$$

Plan

Finding Small Solutions

Applications to RSA

Conclusion

RSA - Recap

- p, q two (large) primes, private
- $n = p \cdot q$, public
- exponents: e public, d private such that

$$ed \equiv 1 \pmod{lcm(p-1, q-1)}$$

- encryption: $c = m^e \mod n$
- decryption: $m = c^d \mod n$

"Cube" attack

- Assume small e, e.g. $e = 3$.
- Assume small m: $m < N^{1/e}$.

"Cube" attack

- Assume small e, e.g. $e = 3$.
- Assume small m: $m < N^{1/e}$.
- Then: $c = m^e \mod n = ...$

"Cube" attack

- Assume small e, e.g. $e = 3$.
- Assume small m: $m < N^{1/e}$.
- Then: $c = m^e \mod n = ... = m^e$.

"Cube" attack

- Assume small e, e.g. $e = 3$.
- Assume small m: $m < N^{1/e}$.
- Then: $c = m^e \mod n = \ldots = m^e$.
- $m = \sqrt[e]{c}$ (over $\mathbb{Z}$)!

"Cube" attack

- Assume small e, e.g. $e = 3$.
- Assume small m: $m < N^{1/e}$.
- Then: $c = m^e \mod n = ... = m^e$.
- $m = \sqrt[e]{c}$ (over $\mathbb{Z}$)!

Example

- Let $e = 3, N = 1000003$.
- Let $m = 100$, then $c = m^e \mod N = 1000000$.
- Clearly, $m = \sqrt[3]{1000000} = 100$.

"Cube" attack (Stereotyped messages)

- Assume small e, e.g. $e = 3$.
- Assume m is *close to a constant* α: $m = \alpha + m_0, m_0 < N^{1/e}$.
- Example: constant padding:
  "today's secret password is: lllattice".

"Cube" attack (Stereotyped messages)

- Assume small e, e.g. $e = 3$.
- Assume m is *close to a constant* α: $m = \alpha + m_0, m_0 < N^{1/e}$.
- Example: constant padding:
  "today's secret password is: lllattice".
- More generally, $c = L(m_0)^e \mod N$, where $L_i \in \mathbb{Z}_{N_i}[x]$ is a public affine map.

"Cube" attack (Stereotyped messages)

- Assume small e, e.g. $e = 3$.
- Assume m is *close to a constant* α: $m = \alpha + m_0, m_0 < N^{1/e}$.
- Example: constant padding:
  "today's secret password is: lllattice".
- More generally, $c = L(m_0)^e \mod N$, where $L_i \in \mathbb{Z}_{N_i}[x]$ is a public affine map.
- Coppersmith: $L(m_0)^e$ is a degree-e polynomial, $m_0 < N^{\frac{1}{e}}$ is a small root!

"Cube" attack (Stereotyped messages) - Example

- Let $e = 3, N = 2000003$.
- Let $m = 1234567 + 7777777m_0$, where $m_0 = 50$.

"Cube" attack (Stereotyped messages) - Example

- Let $e = 3, N = 2000003$.
- Let $m = 1234567 + 7777777m_0$, where $m_0 = 50$.
- Then $c = m^e \mod N = (1234567 + 7777777m_0)^3 \mod N = 39947$.

"Cube" attack (Stereotyped messages) - Example

- Let $e = 3, N = 2000003$.
- Let $m = 1234567 + 7777777 m_0$, where $m_0 = 50$.
- Then $c = m^e \mod N = (1234567 + 7777777 m_0)^3 \mod N = 39947$.
- We know that $m^e - c \equiv 892450 m_0^3 + 1866122 m_0^2 + 726335 m_0 + 302637 \equiv 0 \pmod N$, $m_0^3 + 1684527 m_0^2 + 1652432 m_0 + 1942344 \equiv 0 \pmod N$.

"Cube" attack (Stereotyped messages) - Example

- Let $e = 3, N = 2000003$.
- Let $m = 1234567 + 7777777m_0$, where $m_0 = 50$.
- Then $c = m^e \mod N = (1234567 + 7777777m_0)^3 \mod N = 39947$.
- We know that $m^e - c \equiv 892450m_0^3 + 1866122m_0^2 + 726335m_0 + 302637 \equiv 0 \pmod N$, $m_0^3 + 1684527m_0^2 + 1652432m_0 + 1942344 \equiv 0 \pmod N$.
- Use Coppersmith's method, get $m_0 = 50$.

Hastad Broadcast Attack - Simple

- Assume small e, e.g. $e = 3$.
- Unrestricted m.

Hastad Broadcast Attack - Simple

- Assume small e, e.g. $e = 3$.
- Unrestricted m.
- Broadcasting scenario:
  the same message m is encrypted under e different modulos $N_1, N_2, \ldots, N_e$.

Hastad Broadcast Attack - Simple

- Assume small e, e.g. $e = 3$.
- Unrestricted m.
- Broadcasting scenario:
  the same message m is encrypted under e different modulos $N_1, N_2, \ldots, N_e$.
- $c_1 = m^e \mod N_1, \ldots, c_e = m^e \mod N_e$.

Hastad Broadcast Attack - Simple

- Assume small e, e.g. $e = 3$.
- Unrestricted m.
- Broadcasting scenario:
  the same message m is encrypted under e different modulos $N_1, N_2, \ldots, N_e$.
- $c_1 = m^e \mod N_1, \ldots, c_e = m^e \mod N_e$.
- CRT: reconstruct C such that $C = m^e \mod N_1 N_2 \ldots N_e$.

Hastad Broadcast Attack - Simple

- Assume small e, e.g. $e = 3$.
- Unrestricted m.
- Broadcasting scenario:
  the same message m is encrypted under e different modulos $N_1, N_2, \ldots, N_e$.
- $c_1 = m^e \mod N_1, \ldots, c_e = m^e \mod N_e$.
- CRT: reconstruct C such that $C = m^e \mod N_1 N_2 \ldots N_e$.
- Note that $m < N_1 \Rightarrow m^e < N_1 N_2 \ldots N_e$.

Hastad Broadcast Attack - Simple

- Assume small e, e.g. $e = 3$.
- Unrestricted m.
- Broadcasting scenario:
  the same message m is encrypted under e different modulos $N_1, N_2, \ldots, N_e$.
- $c_1 = m^e \mod N_1, \ldots, c_e = m^e \mod N_e$.
- CRT: reconstruct C such that $C = m^e \mod N_1 N_2 \ldots N_e$.
- Note that $m < N_1 \Rightarrow m^e < N_1 N_2 \ldots N_e$.
- Again: $m = \sqrt[e]{C}$, over $\mathbb{Z}$.

Hastad Broadcast Attack - Harder

- Assume small e, e.g. $e = 3$.
- Unrestricted m.
- Protected broadcasting:
  the same message m is encrypted under e different modulos $N_1, N_2, \ldots, N_e$,

Hastad Broadcast Attack - Harder

- Assume small e, e.g. $e = 3$.
- Unrestricted m.
- Protected broadcasting:
  the same message m is encrypted under e different modulos $N_1, N_2, \ldots, N_e$,
  but padded differently:
  e.g. $m = (m_0 + 2^{|m_0|} i)$.

Hastad Broadcast Attack - Harder

- Assume small e, e.g. $e = 3$.
- Unrestricted m.
- Protected broadcasting:
  the same message m is encrypted under e different modulos $N_1, N_2, \ldots, N_e$,
  but padded differently:
  e.g. $m = (m_0 + 2^{|m_0|} i)$.
- More generally, $c_i = L_i(m_0)^e \mod N_i$, where $L_i \in \mathbb{Z}_{N_i}[x]$ are public affine maps.

Hastad Broadcast Attack - Harder

- Assume small e, e.g. $e = 3$.
- Unrestricted m.
- Protected broadcasting:
  the same message m is encrypted under e different modulos $N_1, N_2, \ldots, N_e$,
  but padded differently:
  e.g. $m = (m_0 + 2^{|m_0|} i)$.
- More generally, $c_i = L_i(m_0)^e \mod N_i$, where $L_i \in \mathbb{Z}_{N_i}[x]$ are public affine maps.
- Can we break this?

Hastad Broadcast Attack - Harder

- Let $g_i(x) := (L_i(x)^e - c_i) \mod \in \mathbb{Z}_{N_i}[x]$.
- Note $g_i(m_0) \equiv 0 \pmod{N_i}$.

Hastad Broadcast Attack - Harder

- Let $g_i(x) := (L_i(x)^e - c_i) \mod \in \mathbb{Z}_{N_i}[x]$.
- Note $g_i(m_0) \equiv 0 \pmod{N_i}$.
- Step 1 - CRT: find $g \in \mathbb{Z}_{N_1 N_2 \ldots N_e}[x], \deg g = e$ such that

$$g \equiv g_i \pmod{N_i}.$$

Hastad Broadcast Attack - Harder

- Let $g_i(x) := (L_i(x)^e - c_i) \mod \in \mathbb{Z}_{N_i}[x]$.
- Note $g_i(m_0) \equiv 0 \pmod{N_i}$.
- Step 1 - CRT: find $g \in \mathbb{Z}_{N_1 N_2 \dots N_e}[x], \deg g = e$ such that

$$g \equiv g_i \pmod{N_i}.$$

- In particular, $g(m_0) \equiv 0 \pmod{N_1 N_2 \dots N_e}$.

Hastad Broadcast Attack - Harder

- Let $g_i(x) := (L_i(x)^e - c_i) \mod \in \mathbb{Z}_{N_i}[x]$.
- Note $g_i(m_0) \equiv 0 \pmod{N_i}$.
- Step 1 - CRT: find $g \in \mathbb{Z}_{N_1 N_2 \dots N_e}[x], \deg g = e$ such that

$$g \equiv g_i \pmod{N_i}.$$

- In particular, $g(m_0) \equiv 0 \pmod{N_1 N_2 \dots N_e}$.
- How? Simply apply CRT to the coefficients.

Hastad Broadcast Attack - Harder

- Step 2 - Coppersmith method:

Hastad Broadcast Attack - Harder

- Step 2 - Coppersmith method:
- $m_0 < N_1 < (N_1 N_2 \dots N_e)^{\frac{1}{e}}$.

Hastad Broadcast Attack - Harder

- Step 2 - Coppersmith method:
- $m_0 < N_1 < (N_1 N_2 \dots N_e)^{\frac{1}{e}}$.
- $g(m_0) \equiv 0 \pmod{N_1 N_2 \dots N_e}, \deg g = e$.

Hastad Broadcast Attack - Harder

- Step 2 - Coppersmith method:
- $m_0 < N_1 < (N_1 N_2 \ldots N_e)^{\frac{1}{e}}$.
- $g(m_0) \equiv 0 \pmod{N_1 N_2 \ldots N_e}, \deg g = e$.
- $\Rightarrow$ recover m_0!

`sage.rings.polynomial.polynomial_modn_dense_ntl.`**`small_roots`**(*self*, *X=None*, *beta=1.0*, *epsilon=None*, ***kwds*)

Let N be the characteristic of the base ring this polynomial is defined over: `N = self.base_ring().characteristic()`. This method returns small roots of this polynomial modulo some factor b of N with the constraint that $b >= N^\beta$. Small in this context means that if x is a root of f modulo b then $|x| < X$. This X is either provided by the user or the maximum X is chosen such that this algorithm terminates in polynomial time. If X is chosen automatically it is $X = ceil(1/2N^{\beta^2/\delta-\epsilon})$. The algorithm may also return some roots which are larger than X. 'This algorithm' in this context means Coppersmith's algorithm for finding small roots using the LLL algorithm. The implementation of this algorithm follows Alexander May's PhD thesis referenced below.

INPUT:

- `X` – an absolute bound for the root (default: see above)
- `beta` – compute a root mod b where b is a factor of N and $b \geq N^\beta$. (Default: 1.0, so $b = N$.)
- `epsilon` – the parameter ϵ described above. (Default: $\beta/8$)
- `**kwds` – passed through to method `Matrix_integer_dense.LLL()`.

Note on Sagemath!

`sage.rings.polynomial.polynomial_modn_dense_ntl.`**`small_roots`**(*self*, *X=None*, *beta=1.0*, *epsilon=None*, ***kwds*)

Let N be the characteristic of the base ring this polynomial is defined over: `N = self.base_ring().characteristic()`. This method returns small roots of this polynomial modulo some factor b of N with the constraint that $b >= N^\beta$. Small in this context means that if x is a root of f modulo b then $|x| < X$. This X is either provided by the user or the maximum X is chosen such that this algorithm terminates in polynomial time. If X is chosen automatically it is $X = ceil(1/2N^{\beta^2/\delta-\epsilon})$. The algorithm may also return some roots which are larger than X. 'This algorithm' in this context means Coppersmith's algorithm for finding small roots using the LLL algorithm. The implementation of this algorithm follows Alexander May's PhD thesis referenced below.

INPUT:

- `X` – an absolute bound for the root (default: see above)
- `beta` – compute a root mod b where b is a factor of N and $b \geq N^\beta$. (Default: 1.0, so $b = N$.)
- `epsilon` – the parameter ϵ described above. (Default: $\beta/8$)
- `**kwds` – passed through to method `Matrix_integer_dense.LLL()`.

1. note 1: δ is the degree of the polynomial

`sage.rings.polynomial.polynomial_modn_dense_ntl.`**`small_roots`**(*self*, *X=None*, *beta=1.0*, *epsilon=None*, ***kwds*)

Let N be the characteristic of the base ring this polynomial is defined over: `N = self.base_ring().characteristic()`. This method returns small roots of this polynomial modulo some factor b of N with the constraint that $b >= N^\beta$. Small in this context means that if x is a root of f modulo b then $|x| < X$. This X is either provided by the user or the maximum X is chosen such that this algorithm terminates in polynomial time. If X is chosen automatically it is $X = ceil(1/2N^{\beta^2/\delta-\epsilon})$. The algorithm may also return some roots which are larger than X. 'This algorithm' in this context means Coppersmith's algorithm for finding small roots using the LLL algorithm. The implementation of this algorithm follows Alexander May's PhD thesis referenced below.

INPUT:

- `X` – an absolute bound for the root (default: see above)
- `beta` – compute a root mod b where b is a factor of N and $b \geq N^\beta$. (Default: 1.0, so $b = N$.)
- `epsilon` – the parameter ϵ described above. (Default: $\beta/8$)
- `**kwds` – passed through to method `Matrix_integer_dense.LLL()`.

1. note 1: δ is the degree of the polynomial
2. warning 2: $\epsilon = \frac{1}{8}$ by default, and is not adjusted!!! (bug?)

sage.rings.polynomial.polynomial_modn_dense_ntl.**small_roots**(*self, X=None, beta=1.0, epsilon=None, **kwds*)

Let N be the characteristic of the base ring this polynomial is defined over: `N = self.base_ring().characteristic()`. This method returns small roots of this polynomial modulo some factor b of N with the constraint that $b >= N^\beta$. Small in this context means that if x is a root of f modulo b then $|x| < X$. This X is either provided by the user or the maximum X is chosen such that this algorithm terminates in polynomial time. If X is chosen automatically it is $X = ceil(1/2N^{\beta^2/\delta-\epsilon})$. The algorithm may also return some roots which are larger than X. 'This algorithm' in this context means Coppersmith's algorithm for finding small roots using the LLL algorithm. The implementation of this algorithm follows Alexander May's PhD thesis referenced below.

INPUT:

- `X` – an absolute bound for the root (default: see above)
- `beta` – compute a root mod b where b is a factor of N and $b \geq N^\beta$. (Default: 1.0, so $b = N$.)
- `epsilon` – the parameter ϵ described above. (Default: $\beta/8$)
- `**kwds` – passed through to method `Matrix_integer_dense.LLL()`.

1. note 1: δ is the degree of the polynomial
2. warning 2: $\epsilon = \frac{1}{8}$ by default, and is not adjusted!!! (bug?)
3. you get $X = \lceil N^{1/d-1/8}/2 \rceil$.
4. for example, $d = 3 \Rightarrow X = \lceil N^{5/24}/2 \rceil$,
   instead of "expected" $N^{1/3} = N^{8/24}$!
5. need to compute required ϵ manually before calling...

```
from sage.all import *

N = next_prime(10**50)
E = 3
x = PolynomialRing(Zmod(N), names='x').gen()

m0 = 10**12 + 20190607 # secret
X = 2*10**12 # bound

m = 1234567890 * m0 + 112233445566778899OO
c = pow(m, E, N)

poly = (1234567890 * x + 11223344556677889900) ** E - c
poly /= poly.leading_coefficient()

epsilon = RR(1/poly.degree() - log(2*X, N))
if epsilon <= 0:
    print "Too large bound X!"
    quit()

print "epsilon:", "2^%f" % RR(log(epsilon, 2))
for root in poly.small_roots(epsilon=epsilon):
    print "root", root
```

Plan

Finding Small Solutions

Applications to RSA

Conclusion

A good resource by David Wong:

github.com/mimoo/RSA-and-LLL-attacks

- implementation of univariate and bivariate Coppersmith algorithms in Sage (from scratch, using LLL);
- also a survey on lattice-based attacks with a good intro.

Another good resource:

Alexander May's Dissertation