Analysis of Arbitrary Waveform Generation for Voltage Glitches

Stanislav Lyakhov, Vincent Immler
vincent.immler@oregonstate.edu

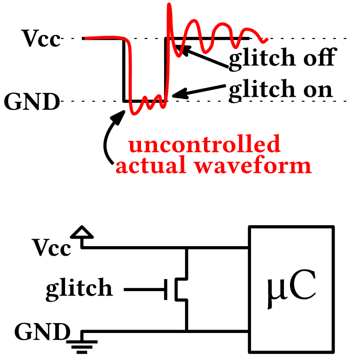
FDTC, September 10, 2023

Oregon State University

Fault Injection (FI) by Voltage Glitching

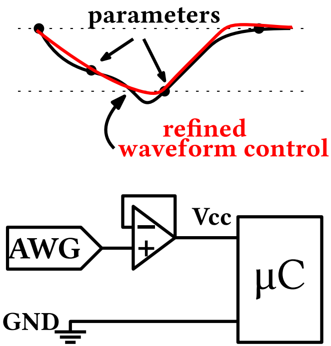
Traditional

square pulse (idealized)



Arbitrary

polynomial interpolation



Why? → Differential Fault Analysis, bypass code read-out protection, fun …

Previous Work

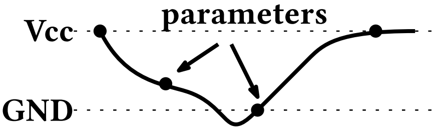
"*Shaping the Glitch*" [1] at CHES 2019

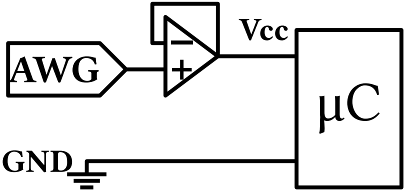
Their approach:

- Cubic interpolation over random points
- Better waveform control vs. traditional glitch
- Shown to improve success rate
- Enabled vulnerabilities in 6 microcontrollers
- Genetic algorithm "discovers" waveform

Limitations:

- High reset rate when glitching (why?)
- Blindly trusting genetic algorithm to do the job
- What features (do not) work in a waveform?





[1] Claudio Bozzato, Riccardo Focardi, and Francesco Palmarini. "Shaping the Glitch: Optimizing Voltage Fault Injection Attacks". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Feb. 28, 2019), pp. 199–224. ISSN: 2569-2925. URL: https://tches.iacr.org/index.php/TCHES/article/view/7390

Our Work and Why it May Matter to You

- **Questions we investigate**
  - How to move away from *random* supporting points?
  - Is it possible to constrain the waveform to hardware-limits?
  - Can we limit the search-space prior to automated learning?
- **Challenges we want to solve**
  - Find properties of a glitch that promote *success* and reduce *reset* rate
  - Generate waveforms more *systematically* while respecting hardware-constraints
  - Provide safety *guarantees* on all generated waveforms
- **Scope of this work**
  - Focus on the properties of the waveform generation prior to automated learning
  - In other words: this is <u>not</u> about the efficiency of the search (*covered separately*)

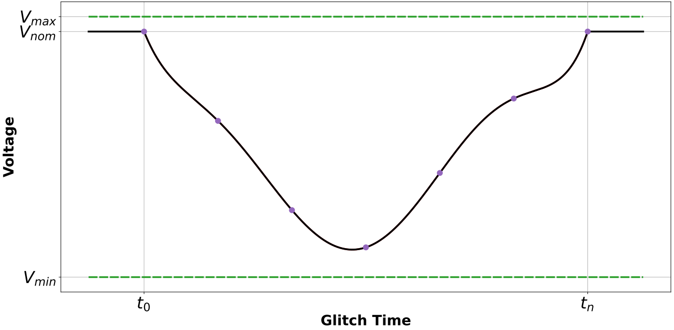
→ targeted improvements important to perform security testing more economically!

Outline of our Work

1. **Waveform Parametrization**
   - Defining “valid” waveforms
   - Polynomial basis: what can go wrong
   - Our proposed approach: *Modular Splines* for generating waveforms
2. **Awgsomefi: Fault Injection Framework (see paper)**
3. **Case Studies**
   - Loop escape on the STM32F0 with *specificity*
   - Improved firmware extraction from the 78K0R
4. **Conclusion**
   - Summary
   - Future Work

Waveform Parametrization: Our Constraints

- Starts at t_0 and ends at t_n
- Begins and ends at V_{nom}
- Must stay within $[V_{min}, V_{max}]$ to avoid damaging the chip

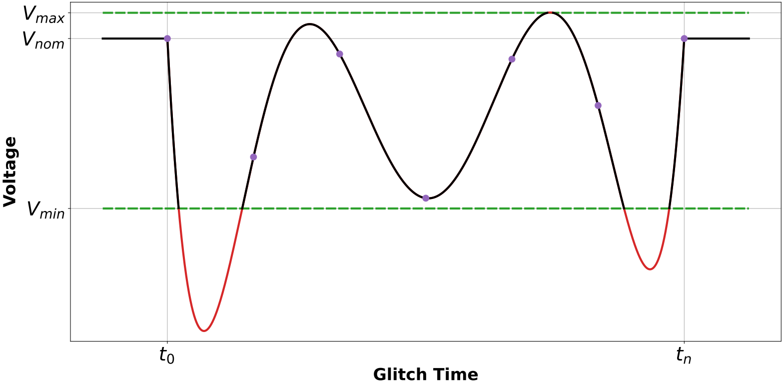


Naive Interpolation

- For simplicity, let's interpolate polynomial on equispaced points
- We set $\mathrm{waveform}(t_0) = \mathrm{waveform}(t_n) = V_{nom}$
- The rest randomly picked to be within $[V_{min}, V_{max}]$
- What can possibly go wrong? (paper=more explanations, discussing cubic interpolation)

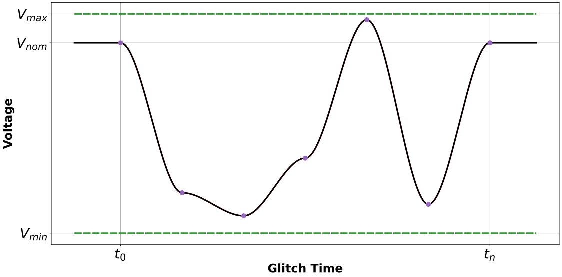
Naive Interpolation

- For simplicity, let's interpolate polynomial on equispaced points
- We set $\mathrm{waveform}(t_0) = \mathrm{waveform}(t_n) = V_{nom}$
- The rest randomly picked to be within $[V_{min}, V_{max}]$
- What can possibly go wrong? (paper=more explanations, discussing cubic interpolation)



Cubic Hermite Splines as Waveforms

- **Needed:** Smoothness/control tradeoff
- Set first derivative at interpolation points
- Interpolation points once differentiable



- **Idea:** What if we set the derivative at each breakpoint to 0?
- Only interpolation points can be minima/maxima in interval…

→ Benefit: waveform will *always* stay inbounds!

Selection of Glitch Duration

- t_0 mostly picked analytically (e.g. by doing side-channel analysis)
- Instead of selecting t_n we look for $t_\Delta = t_n - t_0$
- In previous works this is an additional parameter for exhaustive testing
- **Observation:** Not all durations make sense for complicated waveforms

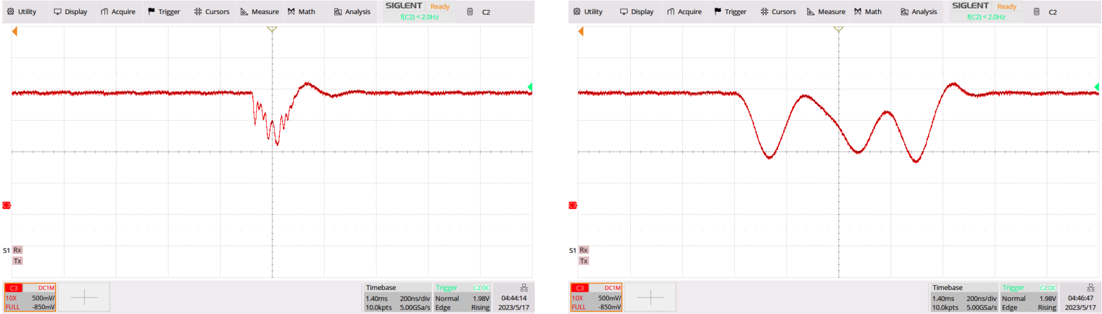


Figure: 120 ns vs. 800 ns glitch.

Modular Splines: Limiting the Slew Rate

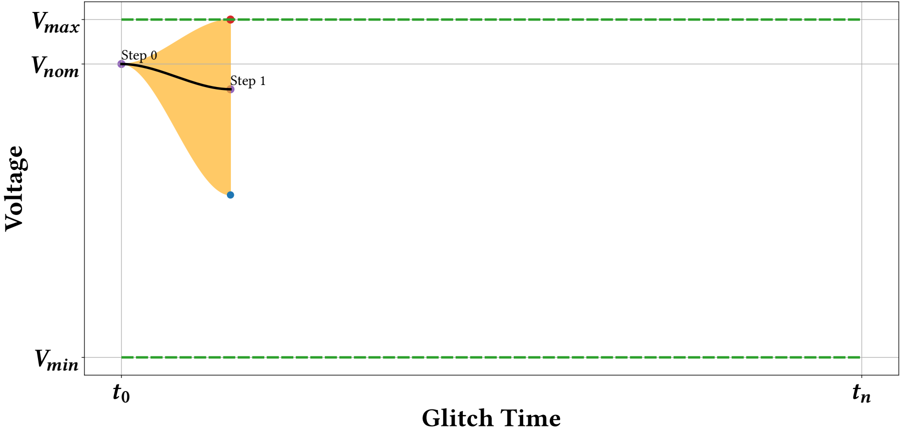
- **Idea:** construct the Hermite spline one segment at a time
  1. Begin at t_0 and $V_0 = V_{nom}$
  2. Choose $t_{i+1} > t_i$
  3. Choose V_{i+1} such that $\left|\frac{V_{i+1}-V_i}{t_{i+1}-t_i}\right| < limit_{\Delta V}$
  4. Repeat from step 2 until $i = n-1$
  5. Choose large enough t_n so that $\left|\frac{V_{nom}-V_{n-1}}{t_n-t_{n-1}}\right| < limit_{\Delta V}$

- For very **small** $t_{i+1} - t_i \implies V_{i+1} \approx V_i$
- For **large** enough $t_{i+1} - t_i \implies V_{i+1}$ chosen from $[V_{min}, V_{max}]$

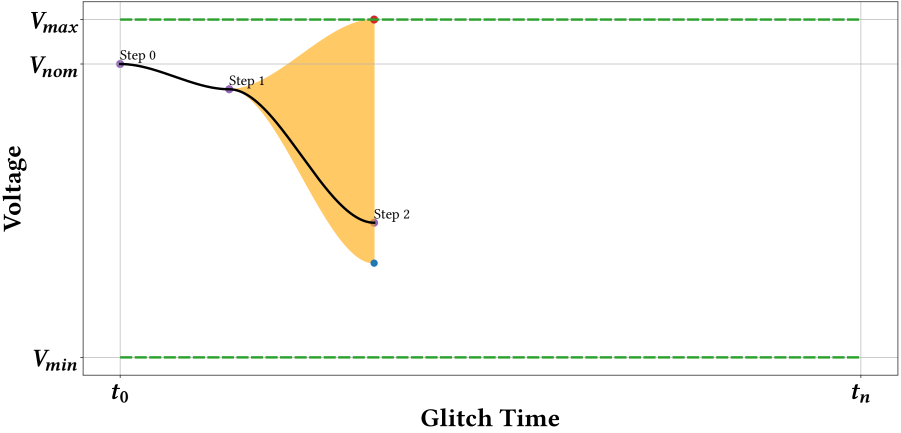
- t_Δ no longer chosen arbitrarily: sum of segments!
- The more "complex" the glitch the higher its average duration

$\rightarrow$ Benefit: search space for t_n determined by OPAMP capability

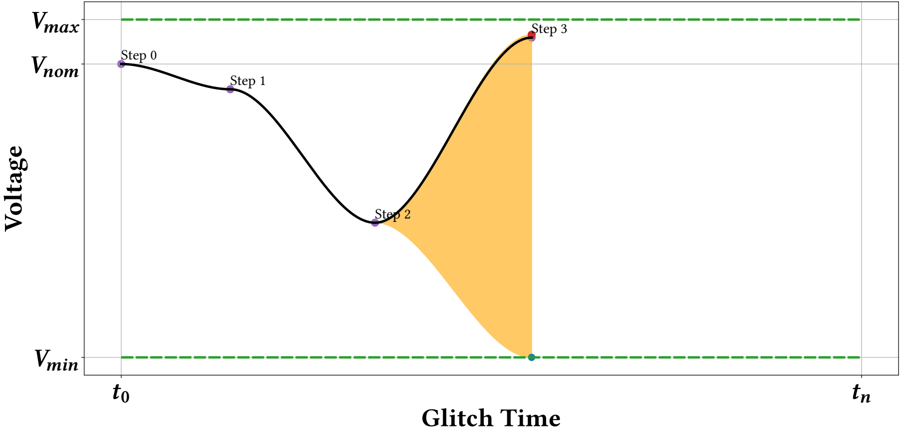
Modular Splines in Action



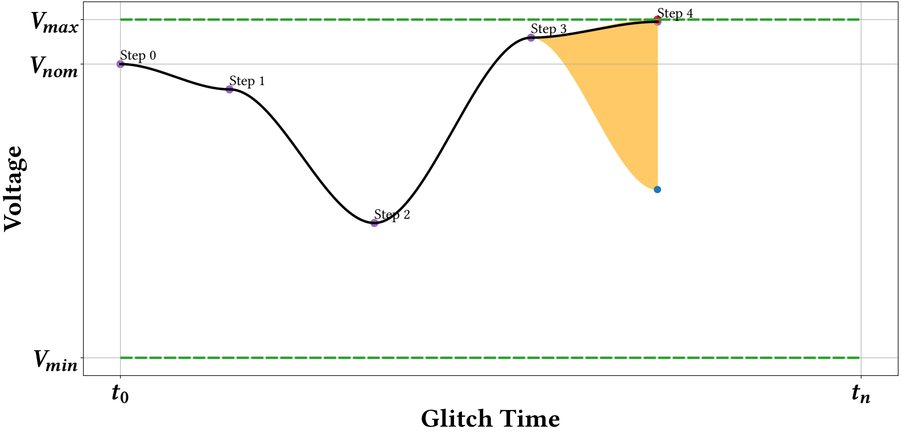
Modular Splines in Action



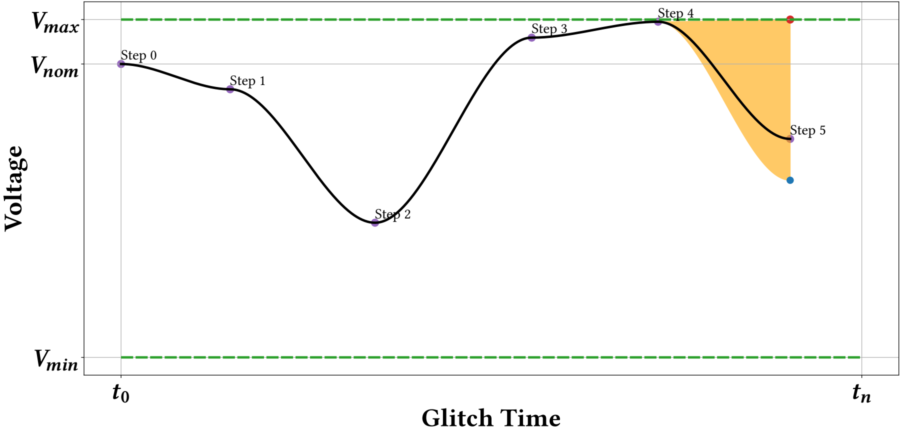
Modular Splines in Action



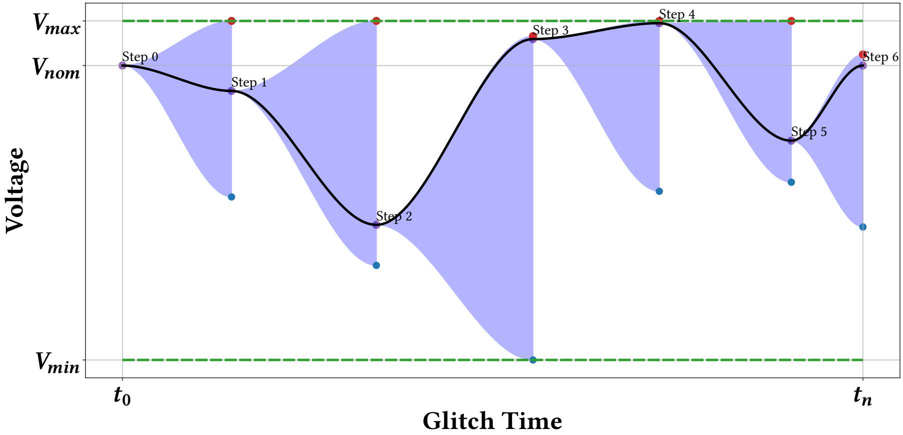
Modular Splines in Action



Modular Splines in Action



Modular Splines in Action



Modular Splines: Summary

- Proposed option is the only one that is *guaranteed* to be in bounds
- Searchspace reduced by bounding voltages at interpolation points based on slew rate

- Full parameters list:
  1. Start time of glitch, t_0
  2. Total number of spline segments, n
  3. Segments t_i (n total)
  4. Inner voltages, V_i ($n-1$ total) bounded by slew rate!

$\longrightarrow$ Modular Splines prove a much more systematic approach with smaller search-space!

Outline of this Work

1. **Waveform Parametrization**
   - Defining “valid” waveforms
   - Polynomial basis: what can go wrong
   - *Modular Splines* for generating waveforms
2. **Awgsomefi: Fault Injection Framework (see paper)**
3. **Case Studies**
   - Loop escape on the STM32F0 with *specificity*
   - Improved firmware extraction from the 78K0R
4. **Conclusion**
   - Summary
   - Future Work

Outline of this Work

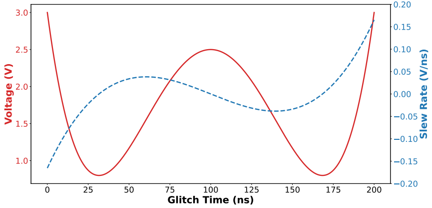
1. **Waveform Parametrization**
   - Defining “valid” waveforms
   - Polynomial basis: what can go wrong
   - *Modular Splines* for generating waveforms
2. **Awgsomefi: Fault Injection Framework** (see paper)
3. **Case Studies**
   - Loop escape on the STM32F0 with *specificity*
   - Improved firmware extraction from the 78K0R
4. **Conclusion**
   - Summary
   - Future Work

Case Study: STM32 Loop Escape

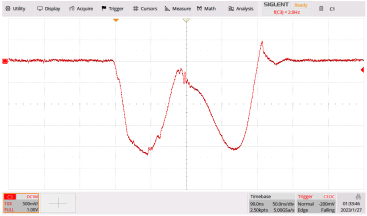
```
while (glitched) // <- Glitch
{
;
}
glitched++;
asm("NOP");
asm("NOP");
// continue NOP slide

if (glitched == 0x01u) {
   sigTermOne();
   sigEnabTwo();
} else if (glitched == 0x02u) {
   sigTermOne();
   sigTermTwo();
}
// Check other possible signals
```

- Goal: escape from loop
- How many instructions skipped?
- Can we control number of skips?

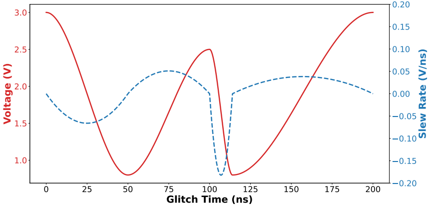
STM32: Waveform Base Comparison



(a) Chebyshev polynomial and its derivative.



(b) Polynomial-glitch applied to STM32.



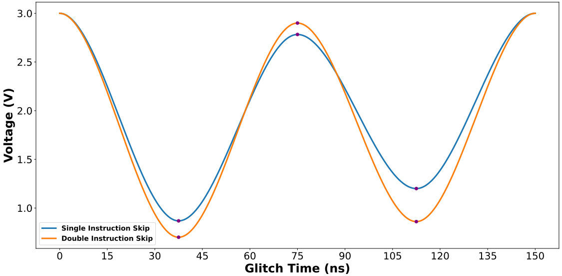
(c) Spline glitch waveform and its derivative.

(d) Spline-glitch applied to STM32.

Figure: Polynomial (0% glitch success) vs. Hermite spline (40% glitch success).

STM32: Exploring Specificity

- Can we look for waveforms that achieve single vs double instruction skips?
- Small difference in waveform: big difference in outcome!



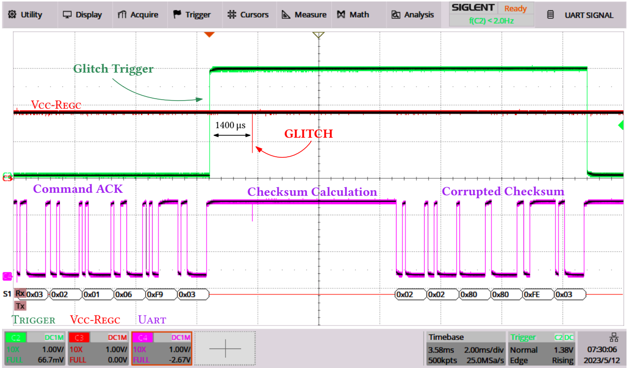
	Single Skip Success Rate	Double Skip Success Rate
Single Skip Waveform	55%	$\approx 0\%$
Double Skip Waveform	6%	41%

Case Study: 78K0R Firmware Extraction

78K0R Plan of Attack: verify-only microcontroller

Need `ChecksumLeak` and `ShortVerify` as gadgets

- `ChecksumLeak`: Omit a byte from the checksum
  - Subtract corrupted checksum from correct checksum to leak a byte!
  - We found waveform that can leak one or **two** bytes
  - Noisy gadget $\implies$ need `ShortVerify`



78K0R Plan of Attack: `ShortVerify`

- `ShortVerify`: Verify multiples of 4 bytes
  - Normally can only verify multiples of 256 bytes
  - Verifications must be 256 byte aligned
  - In previous works, fault injection is used
  - In this work: “properly configured software bypass”

Verify Command (as intended)

Programmer — **78K0R**

- VERIFY[STARTADDR][ENDADDR] →
- ← STATUS FRAME
- DATA FRAME[LEN][DATA][ETB/ETX] →
- ← STATUS FRAME (ST1/ST2)

ShortVerify Command (proposed)

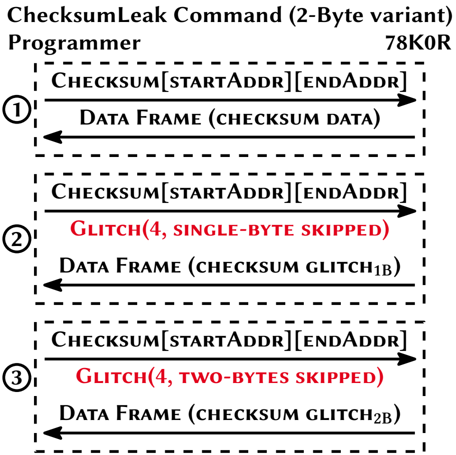
Programmer — **78K0R**

- VERIFY[STARTADDR][STARTADDR+256] →
- ← STATUS FRAME
- DATA FRAME[LEN=4][DATAWORD][ETX] →
- ← STATUS FRAME (ST1/ST2)

Our Attack

Idea: Extract 4 bytes at a time
Let's try to extract bytes $4-7$.

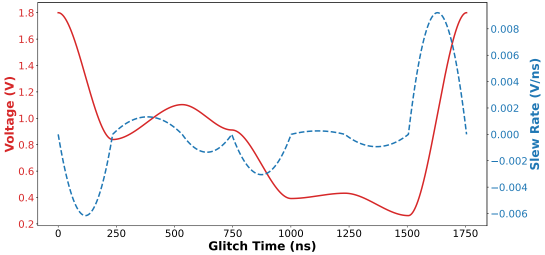
1. Find the reference checksum (no glitch) ①
2. Leak candidates for couplet $4, 5$
   - Use `ChecksumLeak` to omit byte 4 ②
   - Use `ChecksumLeak` again to omit bytes $4, 5$ ③
3. Leak candidates for couplet $6, 7$
   - Use `ChecksumLeak` to omit byte 6 ②
   - Use `ChecksumLeak` again to omit bytes $6, 7$ ③
4. Concatenate candidate couplets and run `ShortVerify` to find the correct bytes!



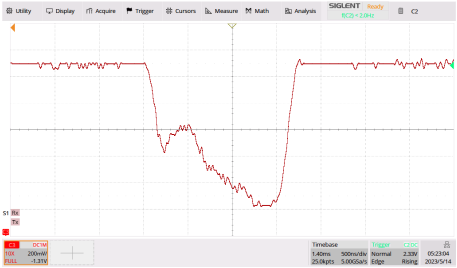
Our Glitch Waveform

- About 2x longer than used in original work
- Allows double glitches

- Smooth: no overshoots
- Lower reset rate $\implies$ more glitches per given time interval



(a) Fault injection waveform.



(b) Scope measurement of fault injection.

Benchmark Comparison

After 10 minutes of glitching (same condition as [1])

Vulnerability	Technique	Success (S)	Reset (R)	R/S	Glitches
ShortVerify	AWG [1]	1291 (6.8%)	2786 (14.6%)	2.16	19044
	this work [2]	**10216 (100%)**	–	–	0
ShortChecksum	AWG [1]	728 (4.4%)	2912 (17.7%)	4.01	16475
	this work [3]	–	–	–	–
ChecksumLeak	AWG [1]	687 (8.6%)	2515 (31.5%)	**3.66**	7977
	this work [4]	1427 (8.8%)	389 (2.4%)	**0.27**	16216

→ Overall, new techniques made this a much more powerful attack!

[2] **software-only bypass: *always* successful!**

[3] **not needed: search-space small enough**

[4] **Reset/Success rate improved from 3.66 to 0.27**

Outline of this Work

1. **Waveform Parametrization**
   - Defining “valid” waveforms
   - Polynomial basis: what can go wrong
   - *Modular Splines* for generating waveforms
2. **Awgsomefi: Fault Injection Framework** (see paper)
3. **Case Studies**
   - Loop escape on the STM32F0 with *specificity*
   - Improved firmware extraction from the 78K0R
4. **Conclusion**
   - Summary
   - Future Work

Summary and Future work

Summary:

- Arbitrary wave voltage glitches provide interesting new opportunities
- Full potential has not been realized yet
- We proposed and demonstrated a new waveform parametrization method
- Using *specificity*, we vastly improved an existing firmware extraction attack

Future work:

- Do we have to interpolate or are there even better options?
- Modular Splines will benefit from new search strategies (work in progress)
- X-device profiling and how well does this perform when facing countermeasures?

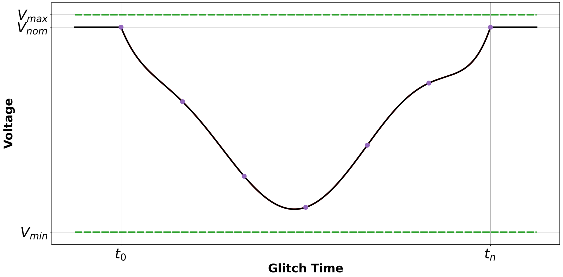
Thank you for your attention!
Questions?

vincent.immler@oregonstate.edu

Backup Slides for Q&A

Arbitrary Waveform Glitches: Polynomials

- **Property 1:** Polynomials can approximate any waveform in the $[t_0, t_n]$ interval
- **Property 2:** Degree n polynomials are *uniquely* determined by $n + 1$ points



- **Idea:** Interpolate polynomials on $n + 1$ points within $[t_0, t_n]$ to generate waveforms
- **Question:** What points t to interpolate on? Does it matter? Yes!

Fixing Oscillations

- **Insight:** even if our interpolation points only pass through $[V_{min}, V_{max}]$, the waveform may still *significantly* oscillate out of bounds
- Why does this happen? Runge Phenomenon!
- Instead we interpolate on *Chebyshev nodes*

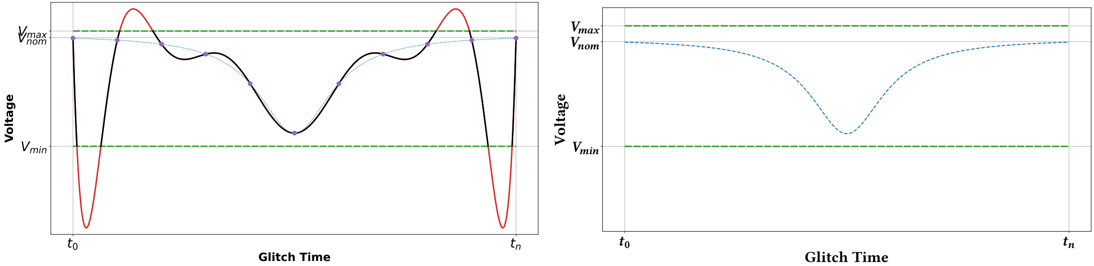
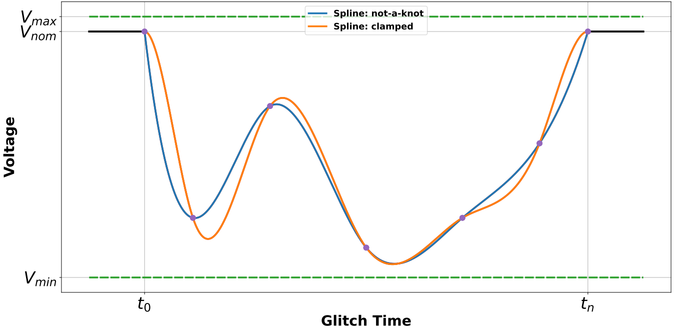


Figure: Interpolating shifted Runge function on equispaced points and Chebyshev nodes.

Cubic Splines as Waveforms

- Used in *all* previous works on arbitrary waveform voltage glitching
- Build complicated waveforms by “stitching” together cubic polynomials
- Spline is twice differentiable at knots + resilient to oscillations!
- Multiple configurations available (clamped, not-a-knot)
- However: slew-rate left uncontrolled!



Cubic Splines: Summary

- Less smooth than polynomials but still twice differentiable everywhere
- Potentially much larger search space (but also better control)
- Stays in bounds (mostly) even on equidistant points