TOPIC: 6.857

DATE: 3/10/21

FILE UNDER:

PAGE: L07.1

Admin:

Pset #2 out

Today:

Cryptographic Hash Fns II

Properties: OW (one-way)
CR (collision-resistant)
.....
TCR (target collision resistant)
PR (pseudorandom)

Applications: Password storage
File modification
Digital signature
Commitment
Voting equipment software
Merkle-Damgard construction & SHA-3
Merkle Trees

Readings:

Katz & Lindell: Ch. 5
Paar & Pelzl: Ch. 11
Ferguson: Ch. 5

Properties: $\mathcal{H} = \{h_s\}$ $h_s(x) = y$ $y \in \{0,1\}^d$

One-way:

Given s, y (where $y = h_s(x)$ for some x)

infeasible to find any x s.t. $h_s(x) = y$

Collision-Resistant:

Given s, infeasible to find any x, x'
s.t. $x \neq x'$
$h_s(x) = h_s(x')$ (collision)

Target Collision Resistance:

Given s, x, infeasible to find any x'
s.t. $x \neq x'$
$h_s(x) = h_s(x')$ (collides with x)

Pseudo-randomness:

Given s, infeasible to distinguish between
- access to h_s
- access to random oracle

Applications:

(1) Password storage

User enters password pw (first time)
System stores $h_s(pw)$ rather than pw

User logins in a second or subsequent tim
enters pw'
System checks that $\underbrace{h_s(pw')}_{\text{computed}} = \underbrace{h_s(pw)}_{\text{stored}}$

Protects against loss of table of hashed passwords
Requires: OW

(2) File modification detector
For each file F, store $(s, h_s(F))$ securely

Later, for file F': check that $h_s(F') = h_s(F)$
(alleged F)

Requires: TCR

(3) Digital signatures

Instead of signing m, sign $h_s(m)$ [efficiency!]

Signer shouldn't be able to find two messages with same hash.

Requires: CR

(4) Commitments

Given a message m, user can compute commitment Com(m, r) to message m. (note randomization r), & give to someone.

Later, user can open commitment by revealing m & r, other party can check.

Com(m, r) should not reveal anything about m.
[Hiding property]

User should not be able to open commitment in more than one way.
[Binding property]

(Also may want non-malleability: can't maul commitment Com(m, r) to get related commitment e.g. Com(m+1, r).)

L07,5

Voting Machine Hashcode Testing: Unsurprisingly insecure, and surprisingly insecure

MARCH 5, 2021 BY ANDREW APPEL 5 COMMENTS

By Andrew Appel and Susan Greenhalgh

The accuracy of a voting machine is dependent on the software that runs it. If that software is corrupted or hacked, it can misreport the votes. There is a common assumption that we can check the legitimacy of the software that is installed by checking a "hash code" and comparing it to the hash code of the authorized software. In practice the scheme is supposed to work like this: Software provided by the voting-machine vendor examines all the installed software in the voting machine, to make sure it's the right stuff.

There are some flaws in this concept: it's hard to find "all the installed software in the voting machine," because modern computers have **many layers *underneath* what you examine**. But mainly, if a hacker can corrupt the vote-tallying software, perhaps they can corrupt the hash-generating function as well, so that whenever you ask the checker "does the voting machine have the right software installed," it will say, "Yes, boss." Or, if the hasher is designed not to say "yes" or "no," but to report the hash of what's installed, it can simply report the hash of what's *supposed* to be there, not what's *actually* there. For that reason, election security experts never put much reliance in this hash-code idea; instead they insist that you can't fully trust what software is installed, so you must achieve election integrity by doing recounts or risk-limiting audits of the paper ballots.

But you might have thought that the hash-code could at least help protect against accidental, nonmalicious errors in configuration. You would be wrong. It turns out that **ES&S** has bugs in their hash-code checker: **if the "reference hashcode" is completely missing, then it'll say "yes, boss, everything is fine" instead of reporting an error.** It's simultaneously ***shocking*** and ***unsurprising*** that ES&S's hashcode checker could contain such a blunder ***and*** that it would go unnoticed by the U.S. Election Assistance Commission's federal certification process. It's unsurprising because testing naturally tends to focus on "does the system work right when used as intended?" Using the system in unintended ways (which is what hackers would do) is not something anyone will notice.

Until somebody ***does*** notice. In this case, it was the State of Texas's voting-machine examiner, Brian Mechler. In **his report dated September 2020** he found this bug in the hash-checking script supplied with the ES&S EVS 6.1.1.0 election system (for the ExpressVote touch-screen BMD, the DS200 in-precinct optical scanner, the DS450 and DS850 high-speed optical scanners, and other related voting machines). (Read Section 7.2 of **Mr. Mechler's report** for details).

We can't know whether that bug was intentional or not. Either way, it's certainly convenient for ES&S, because it's one less hassle when installing firmware upgrades. (Of course, it's one less hassle for potential hackers, too.)

Another gem in Mr. Mechler's report is in Section 7.1, in which he reveals that *acceptance testing* of voting systems is done by the *vendor*, not by the customer. Acceptance testing is the process by which a customer checks a delivered product to make sure it satisfies requirements. To have the vendor do acceptance testing pretty much defeats the purpose.

When the Texas Secretary of State learned that their vendor was doing the acceptance testing themselves, the SoS's Election Division took an action "to work with ES&S and their Texas customers to better define their roles and responsibilities with respect to acceptance testing," according to the report. They may encounter a problem, though: the ES&S sales contract specifies that ES&S must perform the acceptance testing, or they **will void your warranty (see clause 7b)**.

There's another item in Mr. Mechler's report, Section 7.3. The U.S. Election Assistance Commission requires that **"The vendor shall have a process to verify that the correct software is loaded, that there is no unauthorized software, and that voting system software on voting equipment has not been modified, using the reference information from the [National Software Reference Library] or from a State designated repository. The process used to verify software should be possible to perform without using software installed on the voting system."** This requirement is usually interpreted to mean, "check the hash code of the installed software against the reference hash code held by the EAC or the State."

But ES&S's hash-checker doesn't do that at all. Instead, ES&S instructs its techs to create some "golden" hashes from the first installation, then subsequently check the hash code against these. So whatever software was first installed gets to be "golden", regardless of whether it's been approved by the EAC or by the State of Texas. This design decision was probably

a convenient shortcut by engineers at ES&S, but it directly violates the EAC's rules for how hash-checking is supposed to work.

So, what have we learned?

We already knew that hash codes can't protect against hackers who install vote-stealing software, because the hackers can also install software that lies about the hash code. But now we've learned that hash codes are even more useless than we might have thought. This voting-machine manufacturer

- has a hash-code checker that erroneously reports a match, even when you forget to tell it what to match against;
- checks the hash against what was first installed, not against the authorized reference that they're supposed to;
- and the vendor insists on running this check itself — not letting the customer do it — otherwise the warranty is voided.

As a bonus we learned that the EAC certifies voting systems without checking if the validation software functions properly.

Are we surprised? You know: *fool me once, shame on you; fool me twice, shame on me.* Every time that we imagine that a voting-machine manufacturer might have sound cybersecurity practices, it turns out that they've taken shortcuts and they've made mistakes. In this, voting-machine manufacturers are no different from any other makers of software. There's lots of insecure software out there made by software engineers who cut corners and don't pay attention to security, and why should we think that voting machines are any different?

So if we want to trust our elections, we should vote on hand-marked paper ballots, counted by optical scanners, and recountable by hand. Those optical scanners are pretty accurate when they haven't been hacked — even the ES&S DS200 — and it's impractical to count all the ballots without them. But we should always check up on the machines by doing random audits of the paper ballots. And those audits should be "strong" enough — that is, use good statistical methods and check *enough* of the ballots — to catch the mistakes that the machines might make, if the machines make mistakes (or are hacked). The technical term for those "strong enough" audits is ***Risk-Limiting Audit***.

Andrew W. Appel is Professor of Computer Science at Princeton University.

Susan Greenhalgh is Senior Advisor on Election Security at **Free Speech For People**.

FILED UNDER: UNCATEGORIZED

Comments

Douglas W. Jones says:
March 5, 2021 at 9:55 am

Regarding the local officials conducting acceptance testing: The fundamental problem with this is that most counties do not have the expertise to do this. Large urban counties with hundreds of precincts should have an election staff that includes people with real expertise. I've been in the election offices in Washington DC, Miami, Phoenix, Cleveland and places like that, and they really do have people on staff with the ability to do such testing. On the other hand, for each large urban county, there are tens of rural counties, many with just a handful of precincts. Those counties generally have no technical expertise in house and must outsource essentially everything technical, and they typically have no expertise to oversee their outsourcing contractor or contractors. They naturally gravitate to a vendor offering "one stop shopping." The acceptance testing issue demonstrates why this is a big mistake, but we need to find an alternative.

Back when I was chairing the Iowa Board of Examiners for Voting Machines and Electronic Voting Systems, I suggested that small counties should for consortia, sharing an election office between enough counties that they could afford staff with real expertise. They do this with things like maintenance depots for trucks and snow plows, why can't they do it for voting machines? My suggestions in this regard fell on deaf ears.

We have 8 states and several territories with populations below a million. The populations of Wyoming and Vermont are each smaller than the Des Moines metro area. It's impolite to wonder if those states have the necessary technical staff at the state level to do a competent job of acceptance testing. The District of Columbia is only slightly larger, but its election office combines both state-level and local election offices in one organization, and they do have significant technical staffing.

Having significant technical staffing does not imply that that staff is up to the job. I was consulting with the DC election office very shortly before Alex Halderman's students hacked the DC Internet Voting system during their public demo. At the same time I was observing the presence of significant technical expertise, he was demonstrating the kinds of things those experts were mismanaging, as he gained control of the Internet routers in the election office and the security cameras in the server room. My impression was that the DC election office was about as good as it gets, while Halderman and his students demonstrate that that really isn't good enough.

- **2012:** J F M A M J J A S O N D
- **2011:** J F M A M J J A S O N D
- **2010:** J F M A M J J A S O N D
- **2009:** J F M A M J J A S O N D
- **2008:** J F M A M J J A S O N D
- **2007:** J F M A M J J A S O N D
- **2006:** J F M A M J J A S O N D
- **2005:** J F M A M J J A S O N D
- **2004:** J F M A M J J A S O N D
- **2003:** J F M A M J J A S O N D
- **2002:** J F M A M J J A S O N D

author log in

L07.6

Michael says:
March 5, 2021 at 12:49 pm

What if the hashing function were itself implemented in hardware? Fundamentally there's no reason to implement a hashing algorithm in software, so you could make a dedicated piece of silicon that would calculate the hash fingerprint of the entire contents of the rewritable store where the actual voting firmware were held, which could itself by connected a fixed, tamper-evident bus; at that point an auditor could push a button, read the resulting hashed bytes straight off a dedicated register, and compare that to a hash of the voting firmware from the manufacturer. A separate dedicated module could do the same thing using a different hashing algorithm; keep adding discrete modules and hashing algorithms until sufficiently comfortable that whatever is on the firmware disk is either the manufacturer's software or someone has supply-side hacked multiple silicon foundries.

There are plenty of excellent reasons not to move to electronic voting, but I'm pretty sure we can establish that the current state of the bytes on some storage medium is identical to some other known and trusted state of bytes without relying on hackable software. Personally my bigger worry would be the auditors and authorities, all of whom are considerably more difficult to secure from corruption.

Reply

David Jefferson says:
March 5, 2021 at 3:01 pm

Excellent article. Thank you very much for publicizing this ludicrous situation.

The article says that Brian Mechler found a bug in the hash checking script, but I presume that does not cover the embedded hash algorithm it calls. Any such algorithm used for calculating the hash values in the National Software Reference Library should be a well-known *cryptographic* hash, but it does not sound like the EAC requires that. It is not clear to me what ES&S actually implemented. Also, the entire hash-checking process should be open source, even if all of the rest of the code is closed. It is hard to find any justification for keeping that short piece of code proprietary, and there is no way to trust the verification process if the hash algorithm code itself is secret.

In the end, as you say, it is crucial not to base our trust in elections on any kind of trusted software, but on strong and robust post-election risk-limiting audit process.

Reply

Susan Greenhalgh says:
March 5, 2021 at 4:13 pm

IIRC the VVSG requires that the algorithm be a specific NIST cryptographic hash. To your second point, the hash script and the hashing process are included in the vendor's Technical Data Package or TDP. And, unsurprisingly, the whole TDG is considered proprietary and is not public.

Reply

Susan Greenhalgh says:
March 9, 2021 at 9:04 am

An interesting addendum – the EAC's negligence here looks even worse when cast against a recommendation and warning that the GAO issued several years ago, specifically warning that the EAC should be defining test parameters and requiring testing of the software validation scripts. The GAO wrote:

"...the EAC has not established procedures and review criteria for evaluating the effectiveness and efficiency of manufacturer-provided voting system comparison tools. The Program Director told us in September 2012 that the commission requires voting system test laboratories to evaluate such tools and ensure they operate as intended by the manufacturer. However, the commission does not require that manufacturers or testing laboratories apply a standard set of evaluation criteria or test procedures to the tools and the commission has not developed any. Consequently, election jurisdictions still lack an independent framework for determining the accuracy, reliability, security, and usability of manufacturers' software verification tools. The absence of both of these elements of a robust software verification program means that state and local jurisdictions still lack the means to effectively and efficiently verify that voting systems used in federal elections are the same as those certified by EAC."

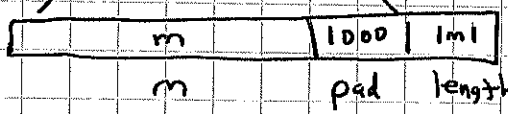
P.S. Thanks David!

Reply

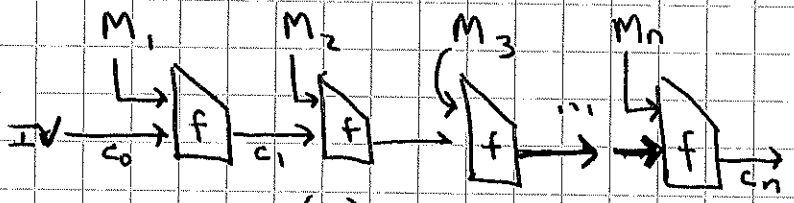
Merkle-Damgard Construction

- Choose output size d (e.g. $d = 256$ bits)
- Choose "chaining variable" size c (e.g. $c = 512$ bits)
  (Make $c \geq 2d$ for good security)
- Choose "message block size" b (e.g. $b = 512$ bits)
- Design "compression function" f
  $f : \{0,1\}^c \times \{0,1\}^b \to \{0,1\}^c$
  [f should be OW, CR, PR, TCR, NM, ...]
- Merkle Damgard is essentially a "mode of operation" allowing for variable-length inputs:
  * Choose a c-bit initialization vector IV, c_0
    (c_0 is fixed & public)
  * [Padding] Given message, append
    - at least one "1" bit, then
    - enough "0" bits so result is a multiple of b bits after concatenation of length of message too.

$M = M_1 M_2 \cdots M_n$ (n b-bit blocks)



Then



$h(m) = c_n$ truncated to d bits

Thm: If f is CR, then so is h.

Pf: Given collision for h, can find one for f by working backwards through chain ▨

Thm: Same for OW.

Common design pattern for f:

$$f(c_{i-1}, M_i) = c_{i-1} \oplus E(M_i, c_{i-1})$$

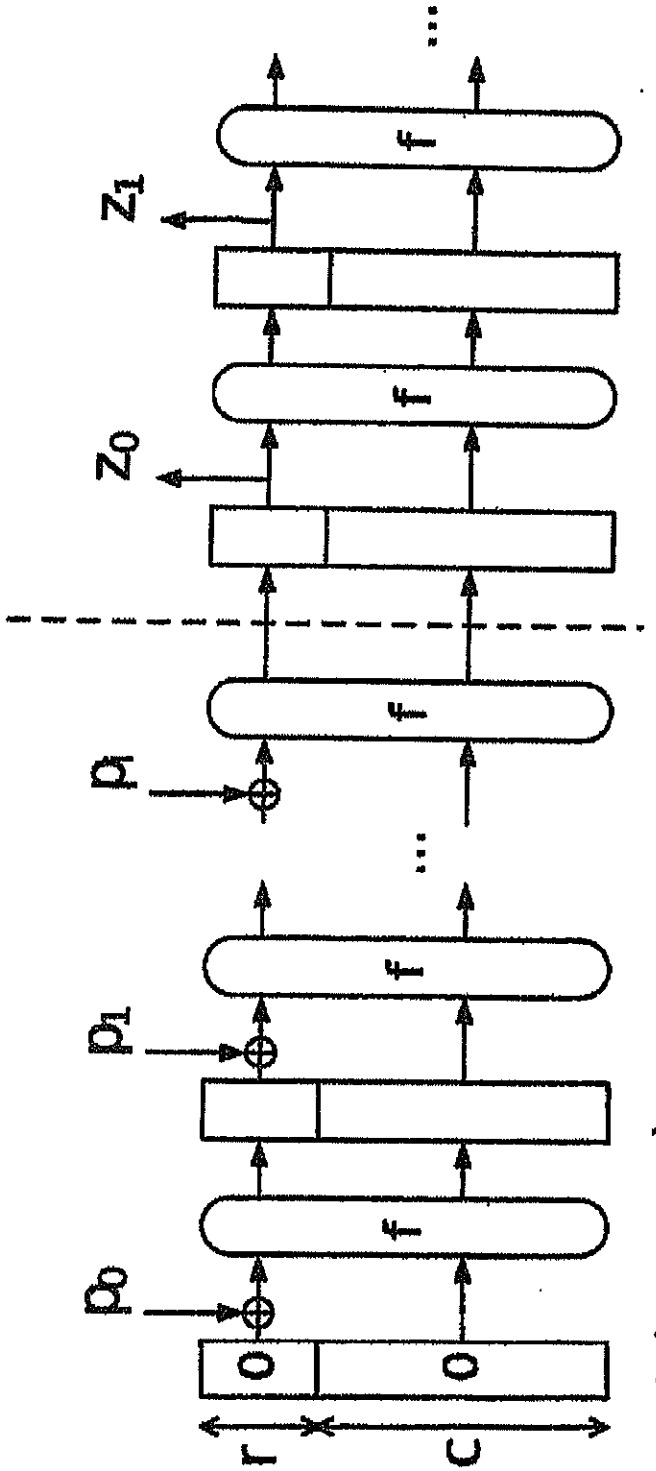
where $E(K, M)$ is an encryption function (block cipher) with b-bit key and c-bit input/output blocks.

(See Wikipedia for more details on MD5.)

SHA-3 (Keccak)

Different construction: "sponge" construction
- Can add entropy (message block) at any time
- Can extract output (squeezesponge) at any time

Keccak = SHA-3



Keccak Sponge Construction

d = output hash size in bits $\in \{224, 256, 384, 512\}$

c = 2d bits

state size = 25 w where w = word size (e.g. w = 64)

c + r = 25w

r ≥ 2d (so hash can be first d bits of Z_0)

Input padded with 10^*1 until length is a multiple of r

f has 24 rounds (for w = 64), not quite identical (round constant)

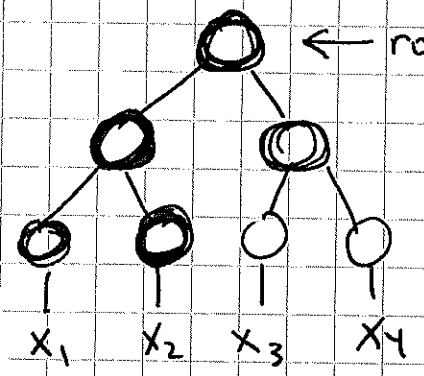
f is public, efficient, invertible function from $\{0,1\}^{25w} \to \{0,1\}^{25w}$

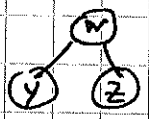
e.g. d = 256
c = 512
r = 1088
w = 64

Merkle Trees

To authenticate not one object, but n objects.

Build a binary tree. Root authenticates all





$w = hash(y \| z)$

hash x_i to get leaf

To authenticate x_i, give all ancestors & all siblings of ancestors.

Need: CR

(Used in bitcoin...)