Vorlesung Nr. 8

Kryptologie II - Datum: 25.10.2018

- Zufall (Teil 2)
- Entropie

Buch der Woche

Titel: Contemporary Cryptology: The Science of Information Integrity (1992)

Autor(en): Gustavus Simmons (Editor)

Verlag: IEEE Press

Umfang: ca. 640 Seiten

Hinweise zum Inhalt:

- uraltes Kryptographie-Lehrbuch, das nach wie vor ein gewisses Alleinstellungsmerkmal aufweist: jedes Kapitel wurde von anderen Kryptographen geschrieben, jede(r) weltweit führend im eigenen Gebiet
- die Themenauswahl ist sehr breit und beinhaltet Gebiete, die in anderen Lehrbüchern zumeist fehlen, beispielsweise Fehler in Protokollen, Kryptoprotokolle zur Überwachung von Abrüstungsvereinbarungen, u.v.m.
- zum Kauf nicht empfohlen, da etwas veraltet; zur Ausleihe aber eine echte Bereicherung
- Tip: Kapitel 3, Whitfield Diffie: The first ten years of public key cryptology

Wahrscheinlichkeitsverteilung

Zufallsbehaftete Prozesse verfügen über eine ***Wahrscheinlichkeitsverteilung***. Jedem Ergebnis eines solchen Prozesses lässt sich eine Wahrscheinlichkeit p zuordnen, wobei $p \in \mathbb{R}$ mit $0 \leq p \leq 1$. Ein unmögliches Ergebnis wird mit p=0 bezeichnet, und ein sicher eintretendes Ereignis mit p=1. Einem Münzwurf ordnen wir üblicherweise $p = \frac{1}{2}$ zu.

Die Summe der möglichen Ergebnisse $p_1, \ldots, p_n$ eines zufallsbehafteten Prozesses ergibt stets 1. Für die Wahrscheinlichkeitsverteilung gilt also: $\sum_{i=1}^{n} p_i = 1$

Sind alle Ereignisse gleich wahrscheinlich, also $p_1 = p_2 = \ldots = p_n$, so sprechen wir von einer ***Gleichverteilung*** (***Uniform distribution***).

Beispiel: Ein idealer Münzwurf verfügt über eine Uniform distribution: $p_1 = p_2 = \frac{1}{2}$.

Beispiel: haben wir einen kryptographischen Schlüssel der Länge n Bit, also beispielsweise n = 128, so ist jeder Schlüssel idealerweise gleich wahrscheinlich, der Schlüsselraum sollte also über eine Gleichverteilung verfügen.

Entropie

Was ist Entropie

Entropie ist eine Maßeinheit für den Informationsgehalt oder, in anderen Definitionen, für das Maß an Ungewissheit, das einem bestimmten Ereignis bzw. einem Sachverhalt zu eigen ist. Entropie lässt sich auf unterschiedliche Weise einführen und definieren, bezeichnet in der Sache jedoch stets das gleiche.

In unserem Kontext interessiert uns die Entropie von Wahrscheinlichkeitsverteilungen $p_1, \ldots, p_n$ binärer Strings. Deren Entropie ist definiert als negative Summe aller Einzelwahrscheinlichkeiten multipliziert mit ihrem binären Logarithmus:

Definition Entropie: $-\sum_{i=1}^{n} p_i \log(p_i)$

Anmerkung: wir verwenden nicht den natürlichen, sondern den binären Logarithmus, da uns dieser diekt den Informationsgehalt in Bits wiedergibt und zudem ganzzahlige Werte liefert, falls unsere p_i Potenzen von 2 darstellen (was hier meistens der Fall ist). Das negative Vorzeichen indes bewirkt, dass das Gesamtergebnis positiv ist.

Entropieberechnung

Schauen wir uns ein paar einfache Beispiele von Wahrscheinlichkeitsverteilungen und deren Entropie an:

Beispiel: angenommen, wir haben einen 128 Bit langen, zufälligen Verschlüsselungs-Schlüssel K = (k_1, …, k_n), n = 128. Dann erhalten wir

$-\sum_{i=1}^{n} k_i \log(k_i) = -\sum_{i=1}^{128} 2^{-128} \log(2^{-128}) = -2^{128} \times 2^{-128} \times (-128) = -1 \times -128 = 128$

Dies entspricht unserer Vorstellung, dass ein solcher Schlüssel insgesamt 128 Bit Information enthält.

Beispiel: für einen idealen Münzwurf mit $p_1 = p_2 = \frac{1}{2}$ erhalten wir $-\sum_{i=1}^{2} \frac{1}{2} \log \frac{1}{2} = 1$.

Wie verhält es sich, würden wir eine verformte Münze verwenden mit

$p_1 = \frac{3}{8}$ und $p_2 = \frac{5}{8}$? Dann gilt (bis auf Rundungsfehler):

$-\sum_{i=1}^{2} p_i \log(p_i) = -\left(\frac{3}{8} \log\left(\frac{3}{8}\right) + \frac{5}{8} \log\left(\frac{5}{8}\right) \right) \approx -\left(\frac{3}{8} \times -1.415 + \frac{5}{8} \times -0.678\right) \approx$

$\approx -(-0.53 - 0.424) \approx 0.954$

Wie wir sehen, sind Informationsgehalt bzw. Ungewissheit geringer geworden. Den höchsten Informationsgehalt erreichen wir bei einer Gleichverteilung.

Erzeugung von (Pseudo-)Zufallszahlen

Erzeugung von (Pseudo-)Zufallszahlen

Wie erzeugen wir nun zufällige oder pseudozufällige Bitfolgen?

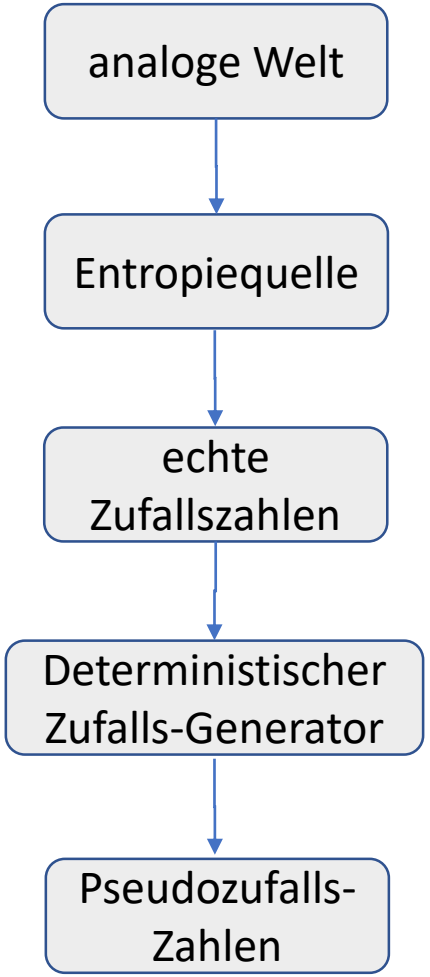
Computer arbeiten rein deterministisch. Versuchen wir daher, ganz ohne echte Zufallswerte eine zufällig aussehende Bitfolge zu erzeugen, so kann diese zwar unter statistischen Gesichtspunkten den gewünschten Anforderungen genügen. Kryptographisch betrachtet fehlt uns jedoch der entscheidende Input, mit dessen Hilfe wir zu verhindern suchen, dass unser Computer vorhersehbare Ergebnisse produziert und Angreifer unsere Bitfolge rekonstruieren.

Für die Erzeugung sicherer kryptographischer Schlüssel benötigen wir zweierlei:

1. eine Entropiequelle in Form echter Zufallswerte: Zufallszahlengenerator bzw. ***Random number generator RNG***
2. ein kryptographisches Verfahren, das auf Basis dieser Entropiequelle Pseudozufallszahlen generiert: ***Pseudorandom number generator PRNG***

In der Praxis wird die Erzeugung “echter” Zufallswerte auf vielfältige Weise vorgenommen, mit unterschiedlicher Qualität.

Echte Zufallsquellen



Für die Bereitstellung echter Zufallswerte eignen sich physikalische Ereignisse wie beispielsweise radioaktive Zerfallsprozesse, sonstige Strahlenemmissionen (Wärme, Akustik etc.), oder andere messbare, analoge Prozesse in beispielsweise Hardwareperipheriekomponenten.

Ersatzweise verwendet man zum Beispiel “Tastaturgeklimper” oder Mausbewegungen des Benutzers (PGP Schlüsselerzeugung), gehashte Speicherinhalte, und vieles mehr.

Moderne PCs verfügen zudem über ein ***Trusted Platform Module TPM***, welches ebenfalls Zufallszahlen liefern kann. Vorsicht: die TPM-Spezifikation lässt hierbei allerdings einige Implementierungdetails offen bzw. variabel, so dass die tatsächliche Güte erzeugter Zufallswerte stark variieren kann…

Aufbau eines PRNG (1)

Ein Pseudozufallszahlengenerator PRNG bezieht regelmäßig “Nachschub” an echten Zufallswerten aus einem Zufallsgenerator RNG. Dieser RHG wiederum erhält seinen Input aus der Messung “zufälliger” analoger Prozesse, wahlweise auch auf Grundlage angeschlossener Peripheriegeräte oder ähnlichem. Der PRNG speichert diese Zufallsbits aus dem RNG in einem Pufferspeicher, dem sogenannten ***Entropy Pool***.

Für die anschließende Erzeugung pseudozufälliger Bits verwendet der PRNG einen deterministischen Algorithmus, der also bei identischen Initialwerten stets dieselben Outputbits liefert. Diesen Algorithmus nennt man daher häufig auch ***Deterministischen Zufallsbitgenerator***, bzw. ***DRBG***.

Die Aufgabe des DRBG besteht darin, aus wenigen echten Zufallsbits viele Pseudozufallsbits zu generieren.

Warum wählt man diese kombinierte Methode? Weil die Systemumgebung normalerweise nicht ausreichend schnell ausreichend viele echte Zufallsbits liefern kann, um einer Anwendung beispielsweise Material zur Erzeugung kryptographischer Schlüssel bereitzustellen.

Aufbau eines PRNG (2)

Ein PRNG verfügt typischerweise über mindestens die folgenden Operationen (deren Bezeichnung abweichen kann):

- init()
- refresh()
- next()

init() initialisiert den Startzustand der Variablen im PRNG, insbesondere den Entropy Pool.

refresh() erneuert oder ergänzt den Inhalt des Entropy Pools mithilfe "frischer" echter Zufallszahlen aus einem RNG. Statt refresh sagt man auch ***reseed***.

next() liefert eine bestimmte Anzahl Output Bits als Pseudozufallswerte

Anmerkung: refresh() wird i.a. vom Betriebssystem aufgerufen, während next() von einer Anwendung aufgerufen wird, die (pseudo-)zufällige Bits benötigt

Anforderungen an PRNG

In vielen Anwendungsszenarien müssen wir davon ausgehen, dass ein potentieller Angreifer Kenntnis erhält über den Output eines PRNG. Damit hieraus kein Schaden entsteht, formuliert man die folgenden Anforderungen:

Backtracking Resistance (auch ***Forward Secrecy*** genannt:

Zuvor erzeugte Bits sollen nicht rekonstruierbar sein.

Dies erreicht man beispielsweise dadurch, dass der Entropy Pool regelmäßig geleert und erneuert wird, da die Kenntnis früherer Entropy Pool Inhalte erlaben würde, den daraus resultierenden Output erneut zu berechnen.

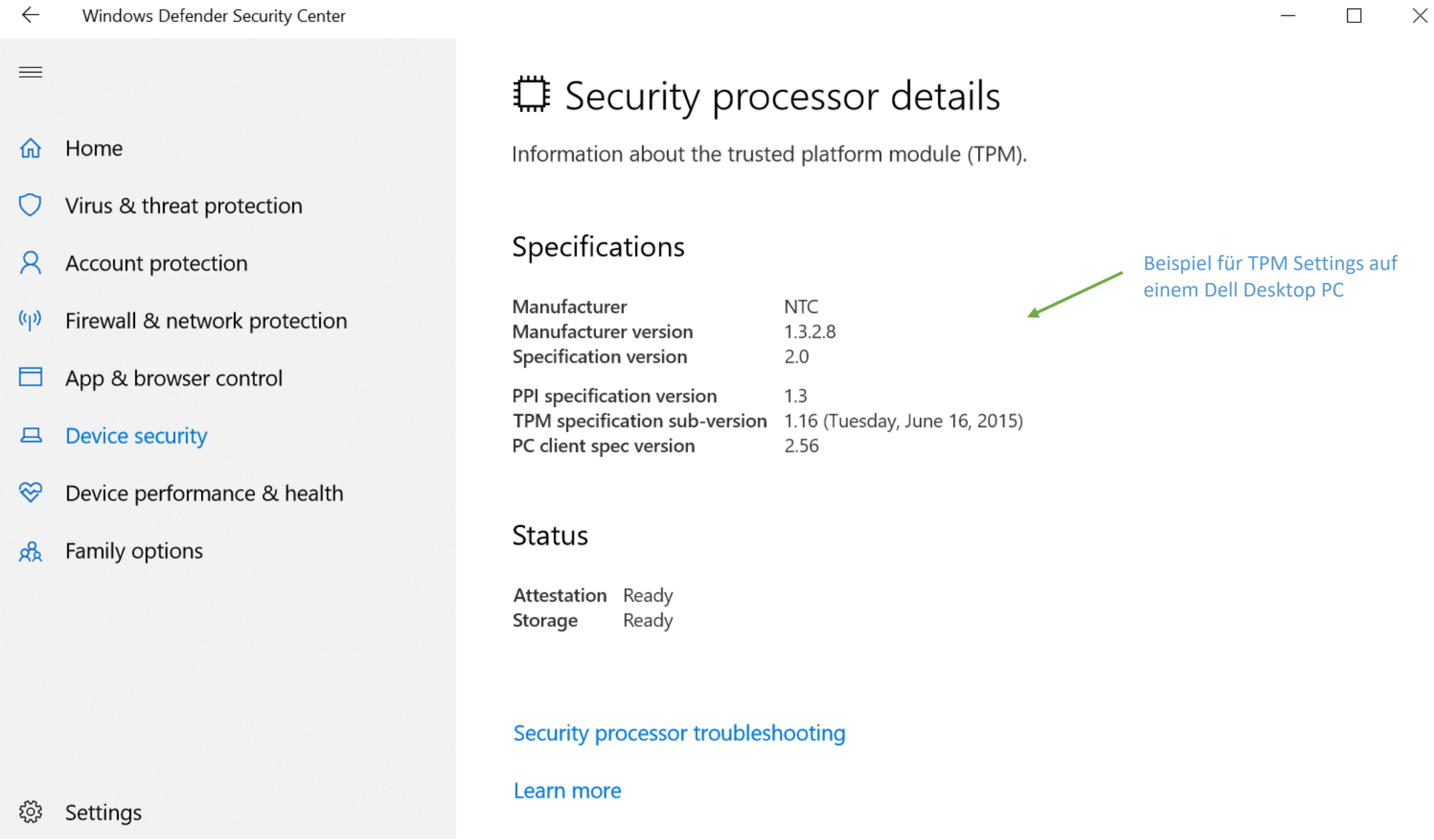
Prediction Resistance (auch ***Backward Secrecy*** genannt):

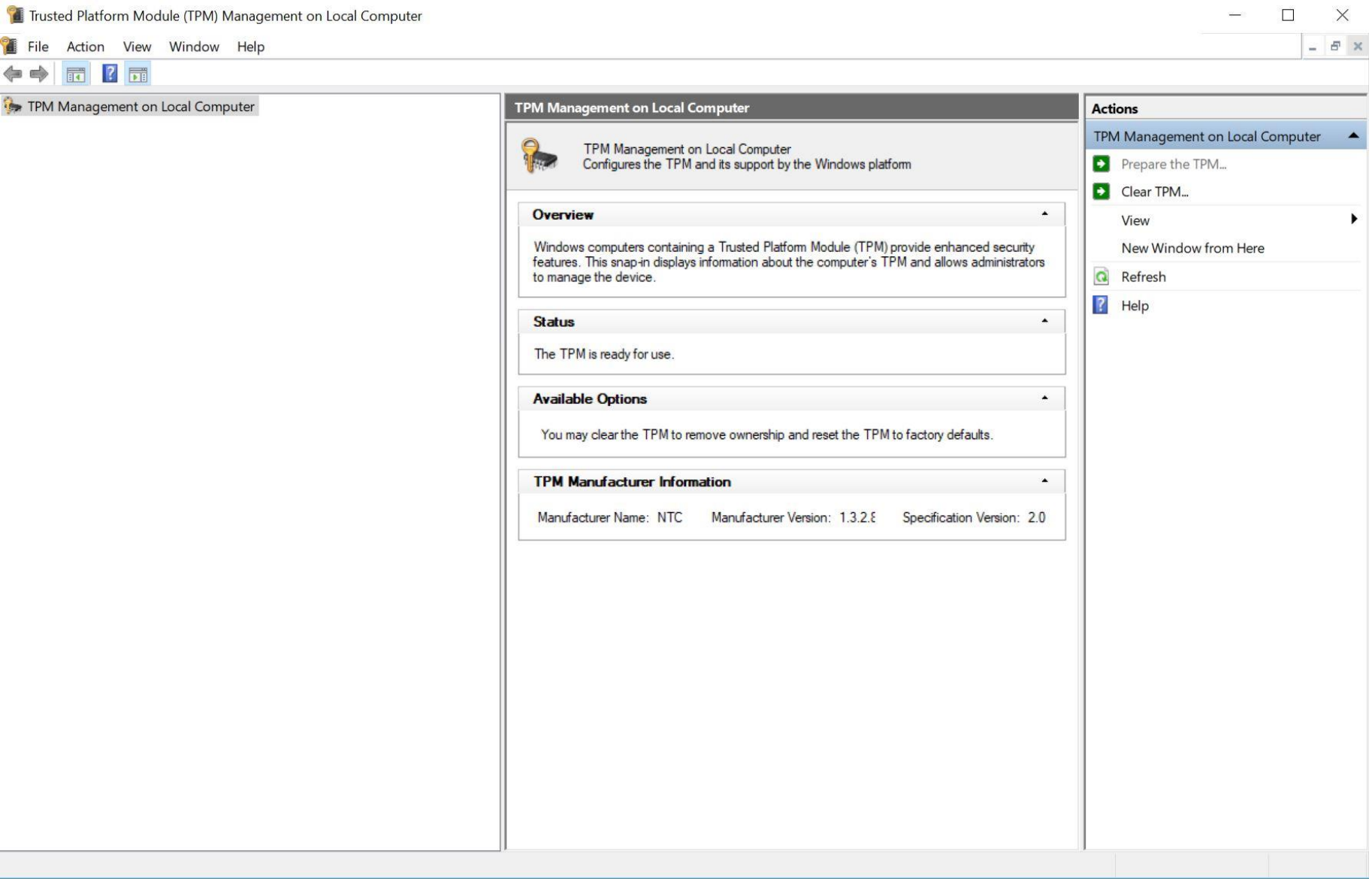
Zukünftig erzeugte Bits sollen nicht vorhersagbar sein.

Dies versucht man zu erreichen, indem man möglichst oft refresh aufruft, den Entropy Pool verändert und dessen Inhalt regelmäßig nachbearbeitet.

Kurzer Exkurs: TPM und Windows

- Microsoft Windows 10 hat eigene Anforderungen an TPM und fordert spezifische Software- bzw. Firmware Releasestände, damit TPM vom Betriebssystem unterstützt werden kann
- Die Dateiverschlüsselung Bitlocker macht Gebrauch von TPM
- Die meisten neueren PCs verfügen über ein TPM. Vor der Nutzung sollte geprüft werden, ob Updates vom jeweiligen Hersteller verfügbar sind, um etwaige Sicherheitsschwächen und/oder Inkompatibilitäten zu eliminieren


Windows Defender Security Center
Home
Virus & threat protection
Account protection
Firewall & network protection
App & browser control
Device security
Device performance & health
Family options
Settings
Security processor details
Information about the trusted platform module (TPM).
Specifications
Manufacturer NTC
Manufacturer version 1.3.2.8
Specification version 2.0
PPI specification version 1.3
TPM specification sub-version 1.16 (Tuesday, June 16, 2015)
PC client spec version 2.56
Beispiel für TPM Settings auf einem Dell Desktop PC
Status
Attestation Ready
Storage Ready
Security processor troubleshooting
Learn more


Trusted Platform Module (TPM) Management on Local Computer
File Action View Window Help
TPM Management on Local Computer
TPM Management on Local Computer
TPM Management on Local Computer
Configures the TPM and its support by the Windows platform
Overview
Windows computers containing a Trusted Platform Module (TPM) provide enhanced security features. This snap-in displays information about the computer's TPM and allows administrators to manage the device.
Status
The TPM is ready for use.
Available Options
You may clear the TPM to remove ownership and reset the TPM to factory defaults.
TPM Manufacturer Information
Manufacturer Name: NTC
Manufacturer Version: 1.3.2.8
Specification Version: 2.0
Actions
TPM Management on Local Computer
Prepare the TPM...
Clear TPM...
View
New Window from Here
Refresh
Help

Nuvoton Technology Corp.
新唐科技股份有限公司

Type	Corporation
Industry	Semiconductor
Founded	2008
Headquarters	Hsinchu Science and Industrial Park, Taiwan
Key people	**Arthur Yu-Cheng Chiao**, Chairman
Products	Microcontroller Application IC, Audio Application IC, Cloud & Computing IC, Foundry Service
Subsidiaries	Nuvoton Electronics Technology (H.K.) Limited Nuvoton Electronics Technology (Shanghai) Limited Nuvoton Electronics Technology (Shenzhen) Limited Nuvoton Technology Corp. America Nuvoton Technology Israel Ltd.
Website	www.nuvoton.com

- Bekannte Hersteller von TPMs sind u.a. Intel, AMD, STMicroelectronics, Infineon und Nuvoton.
- In den Infineon Modulen wurde unlängst eine verheerende Schwachstelle gefunden bei der RSA Key Generation, die ermöglichte, aus Public Keys die Secret Keys zu ermitteln. Die Schwachstelle befand sich mehrere Jahre unentdeckt in einer weit verbreiteten Library, obwohl diese zuvor umfassend evaluiert wurde. Handelte es sich hier um ein Versehen?
- Ein anderer Hersteller, dessen Produkte in unzähligen PCs verbaut werden (z.B. Dell PCs, die tausendfach in westlichen Behörden und Militär eingesetzt werden), ist Nuvoton aus Taiwan, mit mehreren Standorten in China (…)
- Apple PCs wiederum installieren und/oder verwenden überhaupt kein TPM (dies war nur kurze Zeit in 2006 der Fall). Installiert man mittels Bootcamp ein Windows 10 auf einem Apple Computer, so fehlt folglich ein wesentliches Security Feature.

- Anmerkung 1: Apple Mac OSX bietet, analog zu Windows, eine Dateiverschlüsselung an. Was bedeutet das Fehlen eines TPM für deren Sicherheit?
- Anmerkung 2: auch Linux bietet eine Festplattenverschlüsselung, doch im Rahmen einer Standardinstallation keine TPM-Unterstützung. Durch Nachinstallation geeigneter Treiber und Systemeinstellungen lässt sich TPM aber nutzen!

Leere Entropiespeicher

Qualitativ hochwertige RNGs liefern, speziell im PC-Umfeld, nicht immer ausreichend schnell Nachschub zur regelmäßigen Re-Initialisierung eines PRNGs. Dies kann, implementierungsabhängig, unterschiedliche negative Folgen haben.

- Manche PRNG werden geblockt, wenn aus der angeschlossenen Entropiequelle nicht genügend Zufallswerte entnommen werden können. Dies wiederum kann beabsichtigt oder unbeabsichtigt zu Denial of Service führen für die Anwendung, die den PRNG Output benötigt.
- Wird ein PRNG nie geblockt, um ständige Verfügbarkeit zu gewährleisten, so arbeitet der PRNG eventuell mit einem zu geringen Entropie-Input. Dieser sollte, in Bit gerechnet, je nach Anwendung vergleichbar sein mit typischen Schlüssellängen symmetrischer Chiffren, also beispielsweise 128 Bit. Ergeben sich stattdessen z.B. nur 20 Bit, so ist der PRNG-Output nicht mehr hinreichend sicher.
- Zum Systemstart ist der Entropiespeicher oft noch nicht gefüllt und Schlüssel, die kurz danach erzeugt werden, können in der Folge unsicher sein.
- Ein häufiges Praxisszenario sind geklonte Virtuelle Maschinen, bei denen jeder Klon einen Entropiespeicher identischen Inhalts erhält. Da solche Klons nicht mit Kryptoschlüsseln ausgeliefert werden, werden solche Schlüssel häufig unmittelbar nach Start des Klons erzeugt, mit den zuvor genannten Folgen.

Unausgewogene Entropiequellen

Die (oftmals analogen) Quellen, aus denen wir echt zufällige Daten beziehen, produzieren manchmal ein unausgewogenes (engl. **biased**: voreingenommen, parteiisch, verzerrt) Verhältnis von Nullen und Einsen. Zudem kann es Korrelationen geben zwischen einzelnen (oder Teilmengen von) Outputbits. Das macht einen Zufallsgenerator RNG noch nicht unbrauchbar, solange wir eine Möglichkeit haben, dieses Defizit auszugleichen. Bei einem Zufallsbit erwarten wir insbesondere, dass dieses mit Wahrscheinlichkeit $\frac{1}{2}$ wahlweise Null bzw. Eins liefert. Wie können wir unseren Input nachbereiten, damit diese Anforderung erfüllt wird?

1. die analogen Daten lassen sich nachbearbeiten beispielsweise mit Hilfe von Low-pass Filtern. Die Unausgewogenheit lässt sich hiermit zumindest stark reduzieren. Details solcher Verfahren fallen nicht in unser Themengebiet und wir vertiefen diese hier nicht weiter.
2. der analoge Output wird in digitalisierter Form nachbearbeitet. Hierzu stehen verschiedene Methoden zur Auswahl. Man bezeichnet eine solche Nachbearbeitung auch als ***Whitening***.

Von Neumann Extractor

Ein denkbar simples Verfahren zur Eliminierung einer ungleich verteilten Anzahl von Nullen und Einsen wurde bereits 1951 entwickelt von John von Neumann, dem die Informatik zahlreiche bahnbrechende Ergebnisse zu verdanken hat. Das auch ***Von Neumann Extractor*** genannte Verfahren funktioniert folgendermaßen:

Angenommen wir haben eine Folge von Nullen und Einsen. Dann verändern wir diese nach folgendem Verfahren:

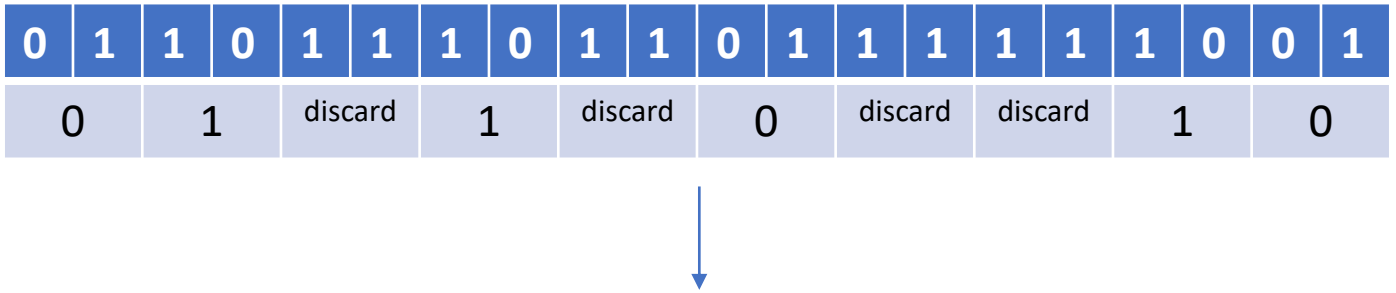
1) wir teilen unseren Bitstrom auf in je zwei aufeinander folgende Bits und betrachten sodann die entstandenen Bitpaare.
2) ein aus zwei identischen Bits 00 oder 11 bestehendes Bitpaar wird verworfen.
3) bei einem aus zwei unterschiedlichen Bits bestehenden Paar behalten wir das erste und verwerfen das zweite Bit.

Man kann zeigen (Details zu technisch hier), dass das Verfahren einen gleichverteilten Output liefert, sofern der Input folgendes erfüllt:

a. es besteht keine Korrelation zwischen je zwei benachbarten Inputbits
b. die Wahrscheinlichkeit dafür, dass ein Inputbit gleich Null ist, ist für jedes Inputbit identisch (wobei sie durchaus verschieden von $\frac{1}{2}$ sein kann).

Von Neumann Extractor: Beispiel

Input: 01101110110111111001



Output: 011010

Inputbits: 6 Nullen und 14 Einsen
Outputbits: 3 Nullen und 3 Einsen

Whitening und Weiterverarbeitung

Der Von Neumann Extractor ist nur eine von vielen Möglichkeiten, wie man den Output einer Entropiequelle bearbeiten kann. Einige weitere Beispiele (es gibt noch andere, hier nicht aufgeführte):

- XOR zweier RNG: bitweise XOR-Verknüpfung des Random number generators (RNG) mit dem Output eines anderen, zweiten Random number generators
- XOR von RNG und CSPRNG: bitweise XOR-Verknüpfung des RNG mit dem Output eines Cryptographically secure pseudorandom number generators (CSPRNG)
- XOR von RNG und symmetrischer Chiffre: bitweise XOR-Verknüpfung des RNG mit dem Output einer Stromchiffre oder Blockchiffre
- RNG und Hashfunktion: der Output des RNG dient als Input einer kryptographischen Hashfunktion; diese fungiert hierdurch als CSPRNG
- RNG als Kryptoschlüssel: der Output des RNG wird verwendet als Schlüssel einer symmetrischen Chiffre. Deren Output dient als CSPRNG

NIST SP 800-90x Anforderungskataloge

Auf der Suche nach technisch ausgereiften Anforderungskatalogen mit hoher Bedeutung für Unternehmen weltweit (und keineswegs nur U.S.-amerikanische Behörden) stoßen wir immer wieder auf das U.S. NIST und dessen hunderte Dokumente rund um das Thema Cybersecurity. Wenig überraschend, finden sich dort auch drei wichtige Dokumente mit Anforderungen an Pseudozufallsgeneratoren:

- NIST SP 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators (January 2012)
- NIST SP 800-90B: Recommendation for the Entropy Sources Used for Random Bit Generation (August 2012)
- NIST SP 800-90C: Recommendation for Random Bit Generator (RBG) Constructions (August 2012)
- NIST SP 800-22: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications (April 2010)

Ebenso empfehlenswert ist RFC 4086: Randomness Requirements for Security (June 2005)

Wie sieht Zufall aus?

Wie sollen wir in der Praxis entscheiden, ob ein Bitstring “zufällig” aussieht oder nicht? Intuitiv würde man fordern, dass keine erkennbaren “nicht zufällig” wirkenden Muster zu erkennen sind. Streng mathematisch und formal erfordert es sehr viel mehr Aufwand, bei einer gegebenen Folge aus Nullen und Einsen ein Urteil zu fällen. Informell stellen wir die folgenden beiden Anforderungen:

1. ***Gleichverteilung*** (***Uniform distribution***): das Auftauchen von Nullen und Einsen innerhalb der Sequenz sollte nahezu gleich sein. Ebenso sollte für jedes Bit die Wahrscheinlichkeit dafür, dass dieses Null ist, gleich $\frac{1}{2}$ sein.
2. ***Unabhängigkeit*** (***Independence***): keine zwei Untermengen unserer Bitfolge sollten in irgendeiner darstellbaren Weise zueinander in Beziehung stehen.

Anmerkung: es gibt verschiedene statistische Tests, denen man eine Bitfolge unterziehen kann, um etwaige Auffälligkeiten bezüglich der oben genannten Forderungen aufzudecken. Forderung (1) ist leicht überprüfbar. Stellt ein Test bezüglich Forderung (2) eine Abhängigkeit fest, so ist ein Negativbeweis erbracht: die Folge ist nicht zufällig.

Die umgekehrte Richtung ist leider nicht möglich: wir können nicht beweisen, dass eine Folge tatsächlich keine Abhängigkeiten enthält.

Was ist eine gute Pseudozufallsfolge?

Wenn wir einen Pseudozufallsgenerator entwerfen oder über dessen Einsatz entscheiden möchten, so brauchen wir Prüfmerkmale und zugehörige Tests für unsere Bewertung. Diese finden wir beispielsweise in oben genanntem NIST SP 800-22. Dort werden insbesondere die folgenden drei Anforderungen gestellt:

1. Gleichverteilung (Uniformity)
2. Skalierbarkeit (Scalability): nicht nur die gesamte gegebene Bitfolge sollte alle Tests bestehen, sondern ebenso jede zufällig erzeugte Teilfolge von Bits.
3. Konsistenz (Consistency): das Verhalten bzw. die Qualität eines PRNG darf nicht davon abhängen, welcher Startwert (Seed) gewählt wurde. Die Tests müssen für beliebige Startwerte erfolgreich ausfallen.

NIST SP 900-22 beschreibt 15 verschiedene Tests für (Pseudo-)Zufallsfolgen. Ein guter PRNG sollte allen genannten Tests genügen.

Anmerkung: Zur Vermeidung etwaig eingebauter Schwachstellen bzw. Hintertüren sollte man den Output eines PRNG im Zweifelsfall nochmals einem weiteren Prozess unterziehen und beispielsweise einen zweiten, anderen PRNG nachschalten.

Next Bit Test

Der sogenannte ***Next Bit Test*** ist ein wesentliches Kriterium zur Bewertung eines kryptographisch sicheren Pseudozufallszahlengenerators CSPRNG.

Angenommen, wir haben eine Folge von k Bits eines CSPRNG. Der Generator erfüllt den Next-bit test, wenn es keinen Algorithmus gibt, der in polynomieller Zeit in der Lage ist, das k+1 te Bit vorherzusagen mit einer Wahrscheinlichkeit signifikant größer $\frac{1}{2}$.

Lassen wir die Feinheiten dieser Definition für den Moment außen vor, so heißt dies schlicht, dass wir zukünftige Bits nicht mit machbarem Rechenaufwand vorhersagen können, auch wenn wir bereits vorhandene Bits kennen.

Mersenne Twister (MT)

Einer der am häufigsten verwendeten PRNG ist der sogenannte ***Mersenne Twister MT***. Dieser kommt in zahlreichen Programmiersprachen (Python, PHP, Ruby, C++, Matlab, Maple, GLib, GNU Multi Precision Arithmetic Library, …) zum Einsatz, ferner in Anwendungen, die Zufallszahlen bereitstellen (z.B. Excel).

Eine Beschreibung des Algorithmus würde etliche Folien beanspruchen, weshalb wir hier darauf verzichten. Wichtig ist indes folgendes:

Mersenne Twister ist kein kryptographisch sicherer PRNG

Somit sollte MT auf keinen Fall zur Verwendung kryptographischer Schlüssel, Nonces etc. verwendet werden.

Begründung: kennt man eine hinreichend lange Folge von Outputbits, so lassen sich alle weiteren Outputs berechnen. Im standardmäßig verwendeten MT genügt hierfür ggf. bereits eine Folge der Länge 624 Bit.

Anmerkung: es finden sich zahlreiche Beispiele für kryptographische Anwendungen, in denen Zufallsgeneratoren wie MT zur Erzeugung von Schlüsseln oder Nonces verwendet werden. Die meisten Entwickler wissen nicht, dass es geeignete und nicht geeignete PRNG gibt.