Journal of Technology Innovations and Energy
https://doi.org/10.56556/jtie.v1i2.134
ISSN: 2957-8809

RESEARCH ARTICLE

Analytical Framework of Cloud Homomorphic Encryption / Cryptographic Logic Obfuscation for Cyber Health Hygiene

Akhigbe-mudu Thursday Ehis[1*]

[1]Africa Institute of Science Administration and Commercial Studies Lome-Togo

Corresponding author: Akhigbe-mudu Thursday Ehis, akhigbe-mudut@iaec-university.tg
Received: 01 January, 2022, Accepted: 02 April, 2022, Published: 15 April, 2022

Abstract
Hacking and the resulting disaster have become so dangerous that developers and organizations are taking extra precautions to reduce their incidence and impact. A drift is any gap between the code and the cloud. Ad hoc adjustments can, of course, result in environment instability, deployment challenges, unpredictably high costs, and security or compliance gaps. The risk of configuration drift becoming a permanent fixture is one of the most important considerations on this study. One such strategic strategy to mitigate these behaviors and render it unavailable to tracking, interpretation, and use by hackers is logic obfuscation. This study presents two strategies for combating piracy and overbuilding attacks. First, the proposed algorithms, then the logic obfuscation develop to hide the functionality and implementation of a design by inserting gates onto the original design. The attackers can use circuit extraction from the gate-level netlist but they won't be able to deduce the obfuscated logic functions. As a result, the proposed obfuscation technique in this brief not only resists image processing but also incurs low area and power overhead.

Keywords: Cloud Encryption; Cryptographic Algorithm; Cyber hygiene; Data at Rest; Homomorphic Encryption, Logic Obfuscation

Introduction

For many individuals and businesses, security is a key consideration when selecting a public cloud provider. Encryption in transit and at rest, which ensures that data can only be accessed by authorized roles and services with proven access to encryption keys, is at the heart of many security strategies. This document appears to capture the fundamental understanding of encryption and cryptographic primitives, which sounds intriguing. Rich theory started to emerge, establishing cryptography as a discipline and a mathematical study. This point of view has influenced how researchers think about computer security in general. Designing and employing codes that permitted two groups to communicate while keeping those messages hidden from eavesdroppers who could monitor communication between them was the old cryptography.
The hard drive acts as a communication channel through which the attacker can listen and read the contents if he or she has access to it. Although "sharing" a key is simple, the user requires a secure and reliable method of remembering / storing the key that will be used over time. In order to write a legal definition, one must consider what is vital in the current crisis and what external structures are necessary (Jonathan and Yehuda, 2020). This section introduces a skewed view of computer privacy as a starting point for a study of current encryption. It demonstrates how this definition may be used to achieve perfect privacy while avoiding the unlikely consequences mentioned earlier, and how a short key can be used to encrypt numerous large communications. An explicit binding of the high chance of success of an enemy running for a certain period of time or, more precisely, investing a specified amount of computational work gauges the security of a cryptographic system. Any definition of security has two parts: a description of what constitutes a system "break" and a description of the adversary's capabilities. Computer secrecy presents two complete secrets: first, confidentiality is guaranteed exclusively to victorious opponents; second, the secret has a small possibility of "failing." This is how to ensure message integrity by detecting any fraudulent messages or disruptions sent over an unprotected communication channel using cryptographic techniques (Gamma et al., 2021). It's easy to believe that encryption addresses the message verification problem. This is for odd, erroneous reasons, such as the opponent being unable to alter the secret message in any logical way because ciphertext totally masks the message content. The message verification code's aim is to prevent the enemy from altering or inserting a new message into a message sent from one person to another unless the recipient notices that the message is not from the intended recipient (Lama Alhathally et al., 2020). What's important to remember is that what defines a good message is entirely dependent on the application. This chapter explores the cryptographic primitive with numerous applications in addition to the existing secure communication problem: Cryptographic hash functions are a type of cryptographic hash function. There are numerous applications for non-conflict hash algorithms expanding the message code verification with

another approach – a standard like Hash Based Message Authentication (HMAC). Between the private writing fields and the public key, hash operations can be seen as a duplicate. Hash functions are increasingly widely utilized in cryptography, and they're frequently used in situations where far stronger structures are required than collision resistance. Hash functions are simple jobs that condense long, unfocused input into short, focused output. In the air, non-collision hash functions are the same; the purpose is to avoid conflict. A explicit collision resistance policy is built into cryptographic hash functions. Because it is impossible to encrypt solid code of every potential key using the necessary amount of memory, hash key functions overcome this technological challenge (Kotsiopoulo et al., 2021).

What is Code Obfuscation?

Hacking and the resulting disasters have become so dangerous that developers and organizations are taking extra precautions to reduce their incidence and impact. The use of code obfuscation is one such strategic strategy that keeps administered codes out of the hands of malicious actors. Code obfuscation comprises remodeling certain characteristics of at-work codes in order to make them inaccessible to tracking, interpretation, and usage by cybercriminals while remaining functional for the developers. The changes to metadata/instructions can be made without affecting the final output that the targeted code provides for application development. Using this method, developers can make the application/program more resistant to hacking attempts. This method performs admirably on all available code kinds.

What is Encryption

Encryption is a process that takes legible data as input (often called plaintext), and transforms it into an output (often called ciphertext) that reveals little or no information about the plaintext. The encryption algorithm used is public, such as the Advanced Encryption Standard (AES), but execution depends on a key, which is kept secret. To decrypt the ciphertext back to its original form, you need to employ the key. At Google,for instance, the use of encryption to keep data confidential is usually combined with integrity protection; someone with access to the ciphertext can neither understand it nor make a modification without knowledge of the key. In this paper, we focus on encryption at rest. By encryption at rest, we mean encryption used to protect data that is stored on a disk (including solid-state drives) or backup media (Sarigiannidis et al., 2021).

What is Data at Rest?

Data at rest refers to data in any digital form that is stored in a computer. This data type is now dormant, as it is not being transmitted between devices or between network points. This information is not actively used by any app, service, tool, third-party, or employee (Panagiotis et al., 2021). At rest isn't a data state that lasts forever. When someone requests a file, the data is transferred across a network and becomes in-transit data. The data enters the in-use state once someone (or something) starts processing it. Both structured and unstructured data can be found in data at rest. The following are some examples of places where a firm can store data at rest: (i) Hard drives and solid-state drives (SSDs) in PCs and laptops. (ii) Database servers. (iii) The cloud, (iv) In a colocation facility operated by a third party and so on. Unlike individual in-motion packets travelling via a network, static data storage often has a logical structure and meaningful file names. Data at rest often contains the company's most sensitive and confidential information, such as: (i) financial documents (past transactions, bank accounts, credit card numbers, etc.). Intellectual property (ii) (product information, business plans, schematics, code, etc.). (iii) Contacts, (iv) Marketing information (user interactions, strategies, directions, leads, etc.). (v) Personal information about employees and customers. (vi) Information on healthcare. Companies frequently replicate files in virtualized environments at rest, backup drives to off-site facilities, allow employees to take laptops home, and communicate data via portable devices, among other things. Data encryption should be used to protect the privacy and security of data at rest. Encryption is the process of translating a piece of data into seemingly meaningless text an unauthorized person (or system) cannot decipher (figure 1).

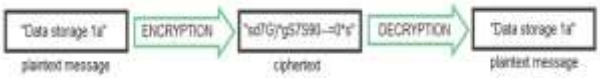


Figure 1: Encryption of Data at rest

Encrypting data at rest is critical for data security, and it reduces the risk of data loss. In most circumstances, symmetric cryptography is used to encrypt data at rest. Unlike asymmetric encryption, where one key scrambles data (public key) and the other decrypts files, the same key encrypts and decrypts the data (private key). When speed and responsiveness are important, as they are with data at rest, security teams commonly use symmetric cryptography. Unfortunately, data encryption isn't just a precautionary measure. At-rest encryption is an important part of cybersecurity since it ensures that data is not easily accessible to hackers. As cybercriminals develop more sophisticated methods for gaining access to and stealing corporate information encrypting data at rest has become a mandatory measure for any security-aware organization (Manuel et al 2018).

What is Cyber Hygiene?

Cyber hygiene refers to the habits and activities that computer and other device users take to keep their systems healthy and secure online (Miamantou et al., 2021). These procedures are frequently followed as part of a routine to protect one's identity and other personal information from

being stolen or tampered with. Cyber hygiene, like physical hygiene, is practiced on a daily basis to protect against natural deterioration and common dangers.

WHAT IS CLOUD ENCRYPTION?

The technique of encoding or modifying data before it is sent to cloud storage is known as cloud encryption (Felix and Isaac, 2021). You're probably already aware that encryption employs mathematical techniques to convert simple data (plaintext), such as a text, file, code, or image, into an unreadable form (ciphertext) that can be hidden from unauthorized or harmful users. It's the simplest and most important technique to ensure that your cloud data isn't hacked, stolen, or viewed by a bad party. Cloud storage companies encrypt data and offer customers with encryption keys. When necessary, these keys are utilized to safely decrypt data. Decryption is the process of converting encrypted data into readable data. There are six distinct areas of the cloud computing environment where equipment and software require significant security attention, as indicated in Figure 2. (Alexandria 2021). These six aspects are: (1) data security at rest, (2) data security in transit, (3) user/application/process authentication, (4) robust data separation between customers, (5) cloud legal and regulatory challenges, and (6) incident response. Cryptographic encryption technologies are unquestionably the finest solutions for safeguarding data at rest. Hard drive manufacturers are currently releasing self-encrypting drives that follow the trustworthy computing group's trusted storage standards (Degabriele and Paterson 2010). These self-encrypting disks have encryption technology, allowing for automated encryption at a low cost and with minimal performance impact. Although software encryption can be used to protect data, it slows down the operation and makes it less secure because an adversary may be able to take the encryption key from the machine without being detected.

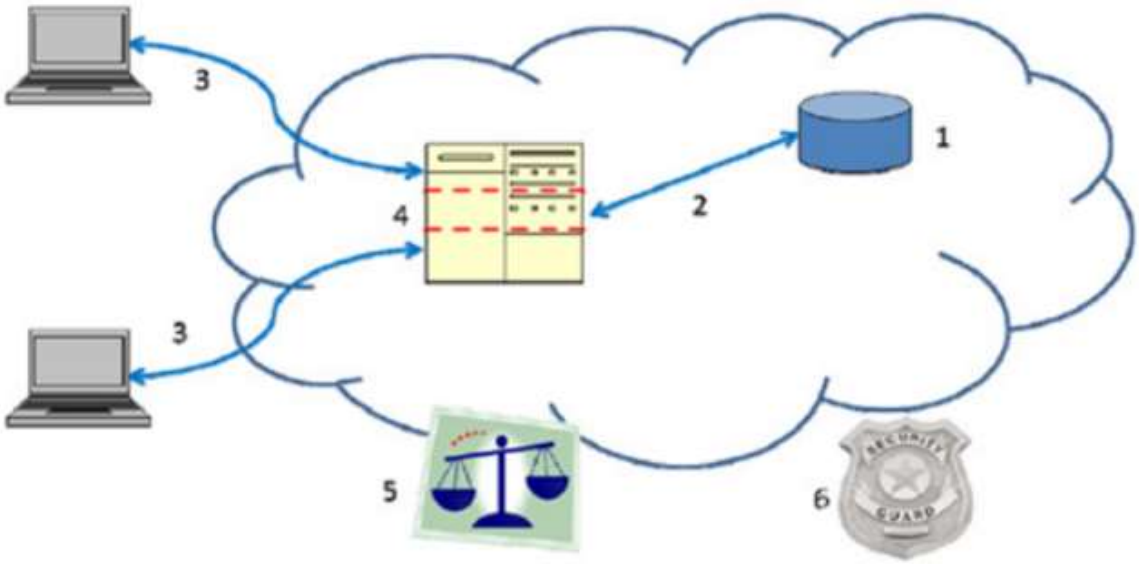


Figure 2: Shows six specific Areas of the Cloud Computing Environment Where equipment and Software Require Security Attention

Statement of the Problem

Most experts agree that encryption is the most effective method for protecting data in the cloud, but it is difficult to implement. For as long as software developers have been writing software, organizations have been working to eliminate misconfigurations in their environments. Configuration drift in the cloud is unavoidable. Even businesses with the best of intentions, who have worked through cloud adoption scenarios and built their infrastructure on well-architected frameworks, frequently encounter deviations from their baseline. When the actual known state of the infrastructure differs from the previous defined configuration, configuration drift happens. Drift can be caused by manually adding or removing resources, as well as making modifications to existing resource definitions. It can also be caused by a variety of automation techniques. Drift isn't inherently bad in and of itself, but given the different severity of effects and the speed of the cloud, awareness is critical, as is having a mechanism in place to resolve unintentional, costly cloud misconfigurations.

Drift can be defined as any code-to-cloud gap, but it doesn't mean all out-of-state configuration changes expose your infrastructure. Ad hoc adjustments can, of course, result in environment instability, deployment challenges, unpredictably high costs, and security in compliance gaps or annoyances to catastrophic production failures.

The risk of drift becoming a permanent fixture is one of the most critical factors on this study. Not understanding why and where a configuration was updated can result in unmanaged blind spots when it comes to evaluating an account's trust boundaries, such as locking its networking

settings or identity-based rights. In a large scale, this can lead to rifts in such borderlines, resulting in catastrophic data breaches and losses. Consider intentionally opening a port to the internet to troubleshoot DNS issues, or providing role administrative privileges to complete a complex database transfer process. There's a chance that those settings established ad hoc using CLI and never reverted after the project or bug was fixed, and they'll become a permanent fixture.

Those are the types of drift that an adversary intending to infiltrate an unprotected cloud asset could take advantage of. On the other hand, not all drifts provide a genuine or exploitable risk; in fact, some drifts may be planned. It's crucial to understand the types of drift that can occur as a result of dynamic infrastructure resource creation or modification to fulfill scalability needs, especially before automating your drift detection.

Literature Review

(Marain et al., 2021) Said that a lot of modern cryptography is now built on solid mathematical underpinnings. However, this does not negate the fact that the field is still a work of art. The rigorous method allows for innovation in establishing definitions that are appropriate for today's applications and surroundings, proposing new mathematical assumptions or inventing new primitives, and constructing and demonstrating secure schemes. Of all, even if cryptosystems are proven secure, there will always be the art of attacking them. Modern cryptography's approach has revolutionized the industry and helps to ensure that cryptographic systems applied in the real world are secure. However, it is critical not to exaggerate what a demonstration of security entails. A proof of security is always conditional on the definition and assumption(s) that are utilized (Mladenav et al., 2020). The evidence may be meaningless if the security promise does not match what is required, or if the threat model fails to represent the adversary's genuine capabilities. Similarly, if the underlying premise proves to be incorrect, the proof of security is useless. The takeaway lesson is that a scheme's proved security does not always imply its security in the real world (Jonathan and Yehuda 2020). While some have seen this as a disadvantage of provable security, we see it as a positive sign of the approach's robustness.

To attack a provably secure scheme in the actual world, all that is required is a focus on the definition (that is, determining how the idealized definition differs from the real-world context in which the scheme is deployed) or the underlying assumptions (i.e., to see whether they hold). Cryptographers, on the other hand, must constantly revise their definitions to make them more realistic, as well as investigate their assumptions to see if they are correct. Provable security does not eliminate the age-old conflict between attacker and defender, but it does provide a framework that helps the defender win.

One of the most often used integrated circuit (IC) protection approaches is logic obfuscation. (Simiosoghlou et al., 2020) Obfuscates IC designs by randomly inserting additional key gates, according to a conventional combinatorial logic obfuscation method proposed (XOR or XNOR). The design's functional input is one of the inputs to a key gate, while the other is a 1-bit key input. To prevent attackers from accessing the proper key, it will be placed in a tamper-evident memory within the design. The obfuscated design will function correctly if you apply the correct key. The IC can be protected from piracy and overbuilding by using logic obfuscation. An opponent could derive the key from the type of inserted gates if the IC was purchased via image processing-based RE (Jonathan and Yehuda 2020).

To circumvent this issue, the authors advocated replacing an XOR gate with an XNOR gate and an inverter, and similarly replacing XNOR gates with XOR gates and inverters, and using de Morgan's law to relocate inversions further up or down. Due to the logic redesign created by de Morgan's rules, this solution has large area and power overheads.

(Mladenov et al. 2020), (Diamantou et al., 2021), and (Yufei and Abhari, 2022) give overviews of a wide range of obfuscation and categorization strategies. On the theoretical side, (Kotsiopuolo et al., 2021) is working on a mathematical model for obfuscation. The most basic strategies substitute equivalent expressions for sections of expressions. Many of the tactics in Hacker's Delight can be useful for obfuscation. Smart equivalences that are difficult to discern are particularly intriguing (Stauch et al., 2022). However, because they are used by popular obfuscators, tools have been developed to precisely remove them (Chaschatzis et al., 2022). Because some algorithms may be identified simply by the presence of constant values (Dimiros et al., 2022), strategies for encoding them have been devised. Mixed-Boolean-Arithmetic and their algorithmic creation are among them (Chausainov and Moscholio 2021), and they look at efficient techniques of constructing various equivalences and encodings that use both logic and arithmetic operations. (Jilian Zhang 2016) suggest opaque predicates to make program analysis more difficult by making it difficult to tell which pathways in a program are taken. Flattening the control flow graph (Radiglou et al., 2021) is another way that has been considered. (Boursiannis et al., 2021) Worked really hard to improve this strategy. The effectiveness of the strategies is assessed using symbolic execution (Felix and Isaac 2021), which is used in the evaluation. It can be used to attack disguised code automatically (Illia Siniosoghu et al., 2021). Not only must implementors be careful to accomplish correctness and speed, but they must also avoid timing and other side-channel leakage.

(Yufei and Abhari 2022) software, it will very certainly struggle with post-quantum cryptographic software's latency. Space and time constraints have always created significant research obstacles for cryptographers, and they will continue to do so for post-quantum cryptographers. On the plus side, cryptography research has yielded numerous impressive speedups, and further research efforts in post-quantum cryptography should continue to yield impressive speedups. Despite substantial cryptanalytic efforts, the (Alexandria, 2021) hash-tree public-key signature system and (Felix and Isaac, 2021) code public-key encryption

scheme were both proposed thirty years ago and remain essentially unchanged. Many other hash-based and code-based cryptography candidates are much more recent; multivariate-quadratic cryptography and lattice-based cryptography provide even more new post-quantum cryptography candidates. There have been several specific suggestions that have been broken. Perhaps a new system will be broken the moment a cryptanalyst takes the time to examine it. One may insist on employing tried-and-true systems that have stood the test of time. However, many users cannot afford traditional systems and must instead examine newer, smaller, faster systems that take advantage of more recent cryptographic technology. To maintain trust in these systems, the community must ensure that cryptanalysts have spent time looking for vulnerabilities. These cryptanalysts, in turn, must learn post-quantum cryptography and get experience with post-quantum cryptanalysis.

The RSA public-key cryptosystem began as a one-way trapdoor function called "cube modulo n." (An interesting historical note: Rivest, Shamir, and Adleman employed huge random exponents in their initial paper (Paisias et al., 2021). Small exponents, such as 3, are hundreds of times faster, as (Radoglou et al., 2021) pointed out.) Unfortunately, a trapdoor one-way function cannot be used in the same way as a safe encryption function. Modern RSA encryption must first randomize and pad a message modulo n before cubing it modulo n (Pliatso et al 2021).

Furthermore, it encrypts a short random string instead of the message to accommodate large communications, and then uses that random string as a key for a symmetric cipher to encrypt and authenticate the original message. It took many years to build the infrastructure around RSA, and there were numerous disasters along the way.

Research Methodology

This paper addresses the fundamentals of cloud computing with its challenge: “Security” in two folds:

(i) It proposes an algorithm for encrypting data at rest before transmitting it for storage in the cloud.

(ii) Logical Obfuscation: Logic Obfuscation hides the functionality and the implementation of a design by inserting additional gates into the original design.

PROPOSED ALGORITHM

The proposed algorithm is to encrypt the data on the client side before transferring it to be stored in the cloud. This will translate the obvious text into ciphertext and prevent data theft by the intruder. That is, even if the attacker is unable to block the data, he will not be able to read the actual data or get a logical explanation from it. The algorithm is equipped with a table of ASCII Codes and Binary Representation codes with zero Padded for easy calculation

Table 1a: ASCII Codes

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

1. Convert the letter to its ASCII code

Encryption Algorithm

2. Convert ASCII code to its 8-bit binary number. If it does not equal 8 bits, add the previous 0s.
3. Find 1 complement to the last 4 pieces.

4. Convert the generated binary code to the ASCII character and transfer it to the cloud.
Example: Suppose we want to send a 'C' over the cloud. First, we convert the plain text 'C' to its ASCII code i.e., 67 uses (table 1). Then we convert 67 to its 8-bit binary number. 67 in binary says 1000011 but since it is not equal to 8 bits, add 1 preceding 0 to get 01000011. This is indicated by (Included Binary Representation Codes No zero). Then we get the completion of 1 of the last 4. This will give us 01001100. Finally, we convert this 8 binary number into its ASCII code letter, 'L'.

B. Decryption Algorithm

1. Get the ASCII code for the character.
2. Convert ASCII code to binary. Add the previous 0s if they are not equal to 8 bits.
3. Subtract the last 4 bits from the 8-digit binary generated.
4. Convert binary generated value to ASCII code.
The real character (blank text) is a letter similar to the ASCII code. Using the example above to convert cipher text into plain text:

First, convert the cipher 'L' text to ASCII code which is 76. 76 and then convert it to binary to get 1001100 but since it does not equal 8 bits, add the previous 0 to get 01001100. Then we undo the last 4 bits. to get 01000011 and convert this binary number to its ASCII equivalent. The original character or plain text is a character similar to the generated ASCII code.

ASCII table

ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so the ASCII code is a numerical representation of a character such as 'a' or '@' or an action of some kind. Below the tables (Tables 1a and 1b) are ASCII characters tables and this includes descriptions of the first 32 unpublished characters. ASCII was actually designed to be used with teletypes so the definitions are unclear. When someone says you want your CV but in ASCII format, all this means you are looking for 'blank' text that has no formatting such as tabs, bold or highlighting - a raw format that can be understood by any computer.

Table 1b: Extended ASCII Codes

128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	▉	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	n
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

Zero-Padded Binary Representation
Convert an integer to a binary string of a specific length in Java

This post will discuss how to convert a number into a binary character unit of a certain length in Java. The solution should attach to the left the binary string with the leading zero. There are several ways to convert a whole number into a binary format in Java:

```
community category NumberFormatUtils {
Public static String longToBinString (val ende) {
  char [] buffer = new character [64];
  Arrays.fill (buffer, '0');
  because (int i = 0; i <64; ++ i) {
    long mask = 1L << i;
    uma ((ival & imaski) == imaski) {
      buffer [63 - i] = '1';
    }
  }
  replace new String (buffer);
```

```
}
public static void main (String ... args) {
   long term =
0b0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
   System.out.println (total);
   System.out.println (Long.toBinaryString (value));
   System.out.println
(NumberFormatUtils.longToBinString (value));
}
}
Output: 5,
101 :
0000000000000000000000000000000000000000000000000000000000000101
```

3.15 (1). Using the String.format () method

To find a binary string of a specific length with leading zeros, we can use the String () class formatting method as follows:

```
Public class Numberformatutils {
Public static string longTobinstring (long val) {
Arrays fill (buffer;'0');
For (int i=0; i<64; ++i){
Long mask =iL<<I;
If (val & Mask )==mask){
Buffer[63-i]='I';
}
```

2. Using StringBuilder:

```
class Main
{
   public static String toBinary(int x, int len)
   {
      StringBuilder result = new StringBuilder();

      for (int i = len - 1; i >= 0 ; i--)
      {
         int mask = 1 << i;
         result.append((x & mask) != 0 ? 1 : 0);
      }
      return result.toString();
   }
   public static void main(String[] args) {
      System.out.println(toBinary(1000, 16));
   }
}
Output: 0000001111101000
```

3.16 (2). Using StringBuilder:

```
class Main
{
   Public Binary Stable (int x, int len)
   {
      StringBuilder Result = New StringBuilder ();
      because (int i = len - 1; i> = 0; i--)
      {
         int mask = 1 << i;
         result.put ((x & mask)! = 0? 1: 0);
      }
      return result.to String ();
   }
   public static void main (String [] args) {
      System.out.println (toBinary (1000, 16));
   }
}
Output: 0000001111101000
```

Drift Detection Manually

The drift detection scan is operational from time to time, depending on the interval set in the description. (Default 1800 seconds, or 30 minutes.) There may be times when you know that files in the directory have changed and you need a summary to be created quickly, but you do not want to change the intermission permanently. Just use manual detection scanning (Figure 3).

1. Click the Inventory tab in the top menu.
2. Search the app.
3. Click the Drift tab.
4. Select the drift definition to use for scanning.
5. Click the Get Now button

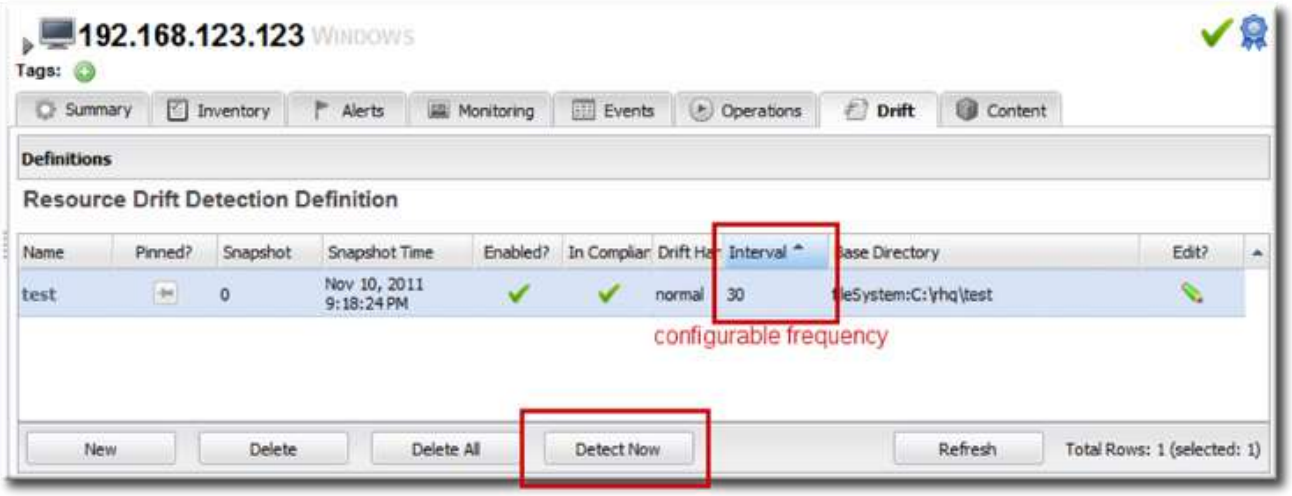


Figure 3: Manual Configuration Drift Detection.

Mathematical Operation Encoding

The simplest method of obfuscation is to integrate mathematical operations by inserting equally strong solid terms that are not visible. Using consistent terminology makes it easy to evaluate the overhead he will appeal to, and by attaching it to unwanted words of the same size, the cost of each shift becomes constant. Because the switch expression will be used instead of the first term, the operating time title is equal to the maximum size and can also be calculated using the above (Figure 4a & 4b). Not all changing words contain not only mathematical words but may also introduce constants, depending on the gross calculation. This may need to be considered in padding. Book (Stauch et al., 2022) provides a rich list of such changes. In Chapter Additive Addition and Logical Performance, many equals of arithmetic integration, subtraction, and denial and all logical operations are clearly presented and proven.

$x + y = x - (\neg y + 1)$

$x + y = (x \vee y) + (x \wedge y)$

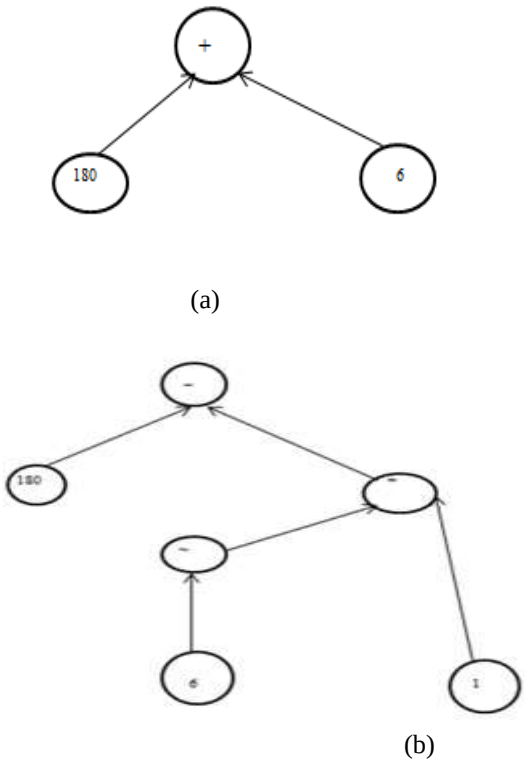
$x + y = (x \oplus y) + (x \wedge y) \bullet 2$

All these changes are the same in that the same complex versions contain both logical and arithmetic functionality. This is common among complex strategies, as it is often difficult to reverse logical expressions and statistics without building a value table. To replicate the same dynamic expressions that complement these mixing conditions can also be found:

$x * y = (x \vee y) \bullet (x \wedge y) + (x \wedge \neg y) \bullet (y \wedge \neg x)$

$x * y = (x \vee y) \bullet (x \wedge y) + \neg (\neg x \vee y) \bullet (x \wedge \neg y)$

The dynamics of these changing expressions depend on the known equations in the analytical tools and development strategies used. By using equilibrium that mixes logical and arithmetic words, this is achieved to some degree. However, because all of these expressions are fixed, this equation can be easily facilitated by using pattern matching methods.



Figures 4a & 4b: Example of Operation Encoding: Encoding of 180 + 6 to the equal Equation $180 - (\neg 6 + 1)$

Logic obfuscation

Logic obfuscation hides performance and application design by adding additional gates to the original build. In order for the design to reflect its correct function (i.e., produce the correct output), a valid key must be assigned to the incomprehensible design. Blurred gates are key gates. When you use the wrong key, the blurred design will show the wrong performance (i.e., the wrong output). Consider the cycle shown in Figure 5 using the " B_1 " and " B_2 " key gates. I0 - I5 input is active and Gate1A and Gate2B are the key connected to the key gates. Using the correct key values (Gate1A = 0 and Gate2B = 1) the design will produce the correct; otherwise, it will produce the wrong output.

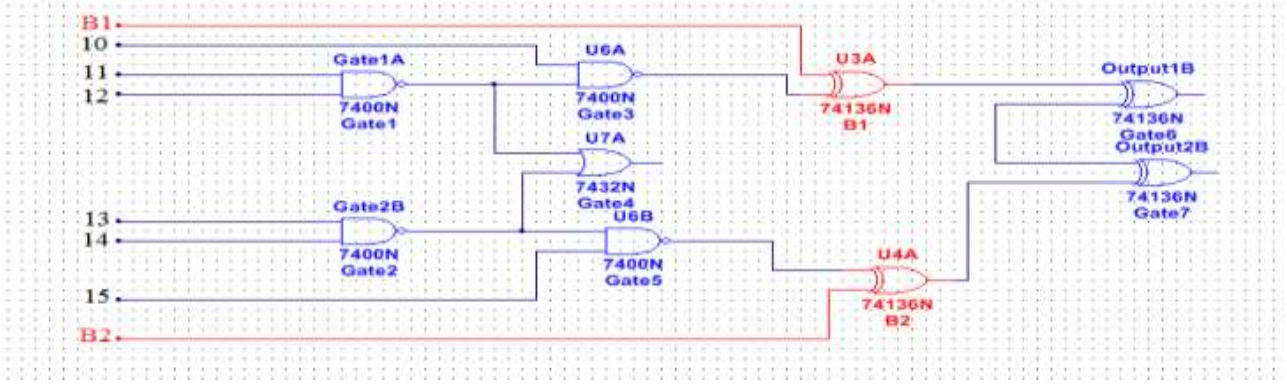


Figure 5: A Circuit Obfuscated Using two keys –gates Gate 1A and Gate 2B, based on the technique proposed (Jilian Zhang, 2016).

By using the 100000 input patterns, the attacker can notify the Gate1A and Gate2B key pieces in Exit B_1 and Output B_2 and view their values.

Definition (attack):

Consider the Gate1A key input in Figure 5. It will be notified on the output of Output1B if the value in one of the G6 input is 0 (uncontrolled gate value OR). This can be achieved by setting input (I1 = 1, I2 = 0 and I3 = 0). As the attacker has access to the active IC, one can use this pattern and determine the value of Gate1A in Output1B. For example, if the Output1B value is 0 in that input pattern, then Gate1A = 0, otherwise Gate 1A = 1. This problem is similar to the problem of reporting errors in the presence of unknown values — X can prevent / hide the spread of error (Alexandria 2021). Key bits Gate 1A and Gate 2B are equal to X sources, as in Figure 5. Similarity and error difference between keystrokes and key distributions: Both purposes require an input pattern that alerts the effect / error key by blocking the result in some or all sources of X / other key fragments, and to prevent their interference.

Result and discussion

Consider the effective cycle illustrated in Figure 5 with two key gates, B_1 and B_2 in different locations. Here, if the attacker has to broadcast the result of any key, then one has to force `0 '(uncontrolled number of gates NOR) in one of the Gate 4 inputs. To force this number, one has to control the main input, which is inaccessible. So one cannot distribute the key effect of the output, failing to determine the key values. Depending on the location of the key gates, different strategies should be used to propagate the key effect. This is the case with a combination of logic obfuscation, as mentioned earlier, when XOR / XNOR gates are introduced to hide design performance (Caspar et al., 2021). Obfuscation is also performed by inserting memory elements (Sariagiannidis et al., 2021). A circuit only works properly if these elements are properly aligned. However, using memory features will add even more important functionality.

Conclusion

In this brief, we have thoroughly analyzed current computer security strategies and developed an effective and efficient way to integrate intelligence to prevent theft, over-construction, and RE attacks. Although attackers may not be able to extract a gate-level netlist with a circuit-based release of retractable engineering tools (RE), they cannot predict obscure logical operations. The only way is to fully evaluate the entire OC configuration with an unreachable brute-force attack. Therefore, our proposed obfuscation process in this brief not only contradicts RE-based image processing but also encapsulates lower space and higher power.

Acknowledgement

Having an idea and turning it into writing is as hard as it sounds. The experience is both internally challenging and rewarding. I especially want to thank the individuals that helped make this happen. Complete thanks to Mrs Josephine Ohens and Mrs Omonigho Akhanolu Casey for their financial support. I will always welcome the chance to represent you.

Thank you to everyone who strives to grow and help others grow. It is the business version of the Lion King Song, “Circle of Life”.

I want to thank God most of all, because without God I wouldn’t be able to do any of this.

Declaration of interests

The author declare that he has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Caspsar Chorus; Sander Van Cranenburgh; Aemiro Melkamu Daniel; Erlend Dancke Sandof’ Anae Sobhani; Teodora Szep (2021): “ Obfuscation Maximization Based Decision Making : Theory, Methodology and First Empirical Evidence”. *Journal of mathematical Social Sciences 109 (2021) 28 - 44.* https://doi.org/10/1016/mathsocsci.2020.10.002

V. Mladenov; V. Chobanov; P. Sarigiannidis; P.I. Radohlou- Grammatikis; Hristov A. and P. Zlatev (2020): “Defense Against Cyber Attacks on the Hydro Power Plant Connected in Parallel with Energy System”. *Jounal (2020 12th Electrical Engineering Faculty Conference, BULEF 2020, Publisher: IEEE*

Siniosoghlou; G. Eftstathopoulos; D. Pliatsios; I.D. Moscholos; A. Sarigiannidis, G. sakellari and Loukas G and P. Sarigiannidis (2020): “ Neuralpot: An Industrial Honeypot Implementation Based on Deep Neural Networks.’ *Proceedings –IEEE Symposium on Computers and Communications.* https://doi.org/10.1109/ISCC.50000.2020.9219712.

Jonathan Katz; Yehuda Lindell (2020); “Introduction to Modern Cryptography. *(3rd ed) chapman and Hall {CRC},* https://doi.org/10.1201/9781351133036

Alexandria V.A (2021): "The Cyber Defense Review, *2019 International Conference on Cyber Conflict U.S { CYCON U.S } Volume 5, No. 1, Spring 2020*

Felix Bentil; Isaac Lartey (2021): "Cloud Cryptography – A security Aspect. International *Journal of Engineering Research and Technology (IJERT) Volume 10, Issue 05, May 2021, pp. 448-450*

Illias Siniosoglou; Panagiotis Rodoglou-Grammatikis and Georgios E; Efstathopoulos; Panagiannids Fouliras and Panagiotis Sarigiannidis (2021): "A Unified Deep Learning Anomaly Detection and Classification Approach for Smart Grid Environments". *IEEE Trabsaction Networks and Service Management (2021), Volume 1, No. 1, 2021,*
https://doi.org/10.1109/TNSM.2021.3078381

T. Kotsiopoulos; P. Sarigiannidis; D. Ioannidis and D. Tzovaras (2021): "Machine Learning and Deep Learning in Smart Manufacturing: The Smart Grid Paradigm. *Journal of Computer Science Review, Volume 40, page 100341*. https://doi.org/10.1016/j.cosrev.2020.100341

I. Siniosoglou ; V. Argyriou; S. Bibi, T. Lagkas and P. Sarigiannidis (2021): " Unsuspervised Ethicak Equity Evaluation of Adversarial Federated Networks." *The 16th International Conference on Availability, Reliability and Security. Pages 1-6*
https://doi.org/10.1145/3465481.3470748

P.Diamantou Lakis ; P. Bouzinis; P. sarigiannidis; Z. Ding and G. Karagiannidis (2021): " Optimal Design and Orchestration of Mobile edge Computing with Energy Awareness. *IEEE Journals of Transaction on Sustainable Computing. Volume 1, No. 1, 2021.*
https://doi.org/10.1109/TSUSC.2021.3103476

Yufei Ding, M. Javadi- abhari (2022): "Quantum and Post-Moore's Law Computing. *IEEE Internet Computing, Vol. 26, pages 5*-6. https://doi.org/1o/10.1109/mic.2021.3133675

M.Stauch; P. Radoglou-Grammatikis; P. Sarigiannidis; G. Lazaridis; A. Orosu; I. Nwankwo; and D. Tzovaras (2022); " Data Protection and Cyber Security Activities and Schemces in the Energy Sector. *Journal of Electronics. Volume 11, No.06*.
https://doi.org/10.3390/electronics.11060965

C. Chaschatzis; C. Karaiskou; E. Mouratidis; E. Karagiannis; P. Sarigiannidis (2022); " Detection and Characterization of Stressed Sweet Cherry Tissues Using Machine Learning. *Journal of Drones. Volume 6, No. 3, (2022), https://doi.org/10.3390/drones6010003*

Dimirios Pliatsios; Sotitios K. Goudos, Thomas Lagkas ; Vasileios Argyriou; Alexandrias Apostolos; A. Boulogeorgos and Panagiotis Sarigiannidis (2022); " Drones Based station for Next Generation Internet of Things; A comparison of Swarm Intelligence Approaches. *IEEE Open Journal of Antennas and Propagation. https://doi.org/10.1109/OJAP.2021.3133459*

I.A Chausainov and I. Moscholios and P. sarigiannidis and M. logothetis (2021): "Multi Service Loss Models for Cloud Radio Access Networks. *IEEE Journal of Access.*
https://doi.org/10.1109/ACCESS.2021.3105946

P. Radoglou- Grammatikis; P. sarigiannidis; E. Iturbe; E. Rios ; S. Martizez ; A. sarigiannidis and G. Eftsathoupoulos and I. Spyridis and A. Seis and N. Vakalis and D. Tzo Varas and E. Kafetzakis and I. Giannidis and M. Tzifas and A. Giannoulakis and M. Tzifas and A. Giannakoulias and M. Tzifas and A Giannakoulis and M. Angelopoupous and F. Tamos (2021): " SPEAR SIEM: A Security Information and Event Management System for the Smart Grid. *Journal of Computer Networks 2021, pages 108008,*
https://doi.org/10.1016/j.comnet.2021.1080008

A.D Boursiannis; M.S. papadopoulou; J. Piaezon; V.C. Mariani; l.S. Coelho and P. Goudos (2021): "Multiband Patch Antenna Design Using Nature Inspired Optimization Method. *IEEE Open journal of Antennas and Propagation, Volume 2, pp. 151 – 152, 2021.*
https://doi.org/10.1109/OJAP.2020.3048490

A. Paisias, T. Kotsiopoulos; G. lazaridis; A.drosu; D. Tzovaras. P. Sariagiannidis (2021): Enabling Cyber- Attacks Mitigation Techniques in a Software Defined Networks. *Proceedings of the 2021 IEEE International Conference on Cyber Security and Resilence. (CSR 2021) PP. 497 – 502. https://doi.org/10.1109/csrs1186.2021.9527932*

D. Pliatsios; P. Sarigiannidis; k. Psannis; S.K. Goudos; V. Vitsas; I. Moscholios (2021): "Big Data Against Security Threats: The SPEAR Intrusion Detection System*," 2020 3rd World symposium on Communication Engineering [WSCE] 2020. Pages 12-17.*
https://doi.org/10.1109/wsce.51339.2020.9275580

Jiliang Zhang (2016): "A Practical Logic Obfuscation Technique for Hardware Security." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 24(3), 1193 - -1198*.
https://doi.org/10.1109/TVLSI.2015.2437996.

J. P. Degabriele and K.G. Paterson (2010). "On the (in) security of IPsec in MAC-then-encrypt configurations". *In 17th ACM Conf. on Computer and Communications Security, pages 493–504. ACM Press, 2010.*

Manuel Cheminod; Luca Durante; Lucia Seno; Adriano Valenzano (2018): "Performance Evaluation and Modelling of an Industrial Applications Layer Firewall". *IEEE Transactions on Industrial Informatics. Volume 14, No.5, pp. 2159-2170. May 2018.*
https://doi.org/10.1109/TII.2018.2802903.

Panagiotis Radoglou-Gammaliks; Panagiotis Sariagiannidis, Eider Iturbe; Erkuden Rios, et al., (2021): Spear Siem: A security Information and Event Management System for the Smart Grid". *Computer Networks".Volume193, July 2021,108008, pages1-26.*

https://doi.org/10.1016/j.comnet.2021.108008

Marcin Wojnakowski; Remiguisz Wisniewski; Grzegorz Bayzydio and Mateusz Poplawskwi (2021): " Analysis of Safeness in a Petri Nets Based Specification of the Control Part of Cyber-physical systems. *International Journal of Applied Mathematics and Computer Science 2021, Volume 31, No.4, 647-657.* *https://doi.org/10.34768/amcs-2021-0045*.

Lama Alhathally; Mohammed A. Alzain; Jchad Al-Amri; Mohammed Baz; Mehedi Masud (2020): Cyber Security Attacks: Exploiting Weaknesses. *International Journal of Recent Technology and Engineering (IJRTE) ISSN:2277-3878, Volume 8, Issues-5, January 2020, pp.906-913.*

Gammal E.I Selim; Ezz El-Din Hemdan; Ahmed M. Shehatta; Nawal A. El-Fishawy (2021): "An Efficient Machine Learning Model for Malicious Activities Recognition in Water-Based Industrial Internet of Things. *Journal Security and Privacy, Volume 4, pp.1-14, Issue 3, May/June 2021.* *https://doi.org/10.1002/spy2.154*.

Frederick Weigang Pan and Matthew Caesar (2016): Salmon: Robust Proxy Distribution for Censorship Circumvention. Proceedings on Privacy Enhancing Technologies". 2016(4): 4-20. https://doi.org/10.1515/popsets-2016-0026.