Word count: 3,242

WEP, Your Time Has Come: Is WEP Safe Enough to

be Anything More Than a Memory?

Supervisor: Mr. Jurcic

Ryan Amos

Since the dawn of wireless communications, privacy has been a major issue. Even before wireless networking, radio transmission often involved disguising your message if the transmission was not meant for public ears. Once wireless networks were created, the issue of privacy invasion soon became prevalent. Credit cards, passwords, and social security numbers were being transmitted via air to the router. What this meant was that it did notrequire a skilled attacker to connect to the wireless network, listen for packets and pull out the sensitive details, or even inject traffic of their own to confuse routers and cause more data to be revealed. In 1997, Wired Equivalent Privacy (WEP) was introduced to the Institute of Electrical and Electronics Engineers (IEEE), which sets the electronic industrial standards for products like wireless security.In response to the development of WEP, the IEEE issued in June of 1997 IEEE 802.11, a set of standards for implementing wireless local area network (WLAN) to improve confidentiality, restrict access, and maintain data integrity.[[1]](#footnote-2)[[2]](#footnote-3) On release, WEP was believed to have data protection standards that matched or exceeded those of wired communications. Before WEP, all wireless network communications were easily accessed. If WEP did its job, a wireless network would allow in no uninvited visitors, reducing identity and data theft as well as protecting the network’s systems from malicious local connections. However, in 2004, the IEEE deprecated WEP because it failed to achieve its goals.WEP was soon replaced with WPA and WPA2, but it still remains very prevalent and is frequently the default encryption method on wireless routers. As such, WEP is still in wide use, in both houses and work environments.[[3]](#footnote-4) Therefore an investigation is needed to determine if these networks should still be using such a system or if networks should begin using better functioning wireless data encryption procedures. Thus the question arises: should WEP be removed from all IEEE 802.11 complying wireless routers? Although it is quite evident that WEP is greatly flawed, WEP still exists. One possible reason is that while WEP is by no means a secure way to transmit credit card numbers, it should be fine for standard internet usage. Breaking WEP is not something everyone can do, so a home network with WEP should be safe from the average neighbor. To determine whether WEP should be removed from wireless routers, one must first examine the structure of WEP, and then examine it for flaws. Finally, one must examine the cost of the flaws and compare them to the benefits.

The reason that WEP was deprecated is because it is so simple to obtain the root key. This is quite evident in the numerous effective techniques for cracking the system. At the time of writing, WEP keys can be determined in seconds once the appropriate data is gathered, which takes nothing more than a few minutes on a busy network. Despite this, WEP is still in use. The option to use WEP to encrypt data is not only offered by many wireless routers, but often the default option.[[4]](#footnote-5) By studying the issues behind WEP, it may be possible to make the case that the IEEE should not only deprecate, but eliminate WEP from all Wi-Fi certified routers. If such an action were to take place, many intrusions may be thwarted. However, one must weigh the risk of intrusion against the benefit of backwards compatibility to determine if WEP should be banned from IEEE 802.11 wireless routers.

Attacks on wireless networks can be quite dangerous to personal privacy, so traffic must be protected in some manner. The most basic solution to the problem of wireless security is the medium access control (MAC) filter, which accepts or rejects clients based on their MAC address, which is a unique address for each network interface card (NIC). However, this does not prevent the attacker from listening to packets―and thus the MAC address of the sender. Once the MAC address has been obtained, it is trivial to change the NIC's MAC address and transmit packets over the network.[[5]](#footnote-6)

Wireless encryption will never be attack-proof. However, through robust encryption techniques, wireless encryption can be so attack-resistant that the probability of an intruder will be negligible. Brute force is a password cracking technique that will work with any encryption algorithm, although as the keyspace, which is the size of the key, increases, the algorithm takes longer to complete: some brute force algorithms can take lifetimes to complete. Therefore, it is necessary to examine WEP to decide if WEP is sufficiently attack-resistant to merit its existence as any more than a relic.

The WEP encryption algorithm has many flaws that make the key obtainable within a few seconds once the necessary packets are collected. To encrypt a packet, a pseudorandom initialization vector (IV), a 24 bit numerical value, is generated.[[6]](#footnote-7) Using the IV and the key, which is shared between the client and router, a keystream of a large length, is generated using the RC4 encryption algorithm.[[7]](#footnote-8) RC4 creates a pseudorandom number, generated from a seed. An RC4 message, *M*, is the message to be transmitted, which is transformed into a plaintext message *P*, not to be confused with the message *M*, by appending the integrity checksum of the message, ; such that . The ciphertext is then produced through an XOR operation.[[8]](#footnote-9) XOR is a binary operation, denoted, which is similar to a not-equals operation.[[9]](#footnote-10)The XOR operation compares two binary values and returns a 1 if they are different and a 0 if they are equal. Combining the RC4 message with XOR decoding yields the ciphertext which is . An example of this encryption algorithm is demonstrated in the code box below, written in Java. To use this algorithm, one would input into the RC4 initialization sequence and feed in the bytes of in sequential order. The output would be the ciphertext. In the example, integers are used instead of unsigned bytes because Java requires signed (positive or negative) values. In the example below, the basic algorithm was provided as referenced, and I created code following the algorithmic description found at http://web.archive.org/web/20080404222417/http://cypherpunks.venona.com/date/1994/09/msg00304.html.

/\*\*

Code

\* RC4 Algorithm source:

\* http://web.archive.org/web/20080404222417/http://cypherpunks.venona.com/da

\* te/1994/09/msg00304.html[[10]](#footnote-11)

\* **@author** Ryan Amos

\*/

**public class** RC4

{

**private int**[] state;

**private int** x;

**private int** y;

**public** RC4(**int**[] key)

{

**int** index1 = 0;

**int** index2 = 0;

x = 0;

y = 0;

state = **newint**[256];

**for** (**int** i = 0; i < state.length; i++)

state[i] = i;

**for** (**int** i = 0; i < state.length; i++)

{

index2 = (key[index1] + state[i] + index2) % 256;

**int** temp = state[index1];

state[index1] = state[index2];

state[index2] = temp;

index1 = (index1 + 1) % key.length;

}

}

**public int** encrypt(**int** b)

{

x = (x + 1) % 256;

y = (state[x] + y) % 256;

**int** temp = state[x];

state[x] = state[y];

state[y] = temp;

**int** xorIdx = (state[x] + state[y]) % 256;

**return** b ^ state[xorIdx];

}

}

As an example, assume the key is “foo” and the message is “bar”. First, the unsigned byte values for “foo” must be obtained. Any map of character to byte, such as ASCII, will work. This array is then fed into the initializer. After the stream has been initialized, each byte of “bar” is fed in and XORed with the stream. The result will be a string composed of characters 120, 215, and 198, or “x×Æ” when mapped to ASCII. The final packet that is transmitted via radio wave is the initialization vector appended to the ciphertext.[[11]](#footnote-12)The whole packet is shown in figure 1.

IV (24 bits)

Ciphertext

Final packet

RC4

IV (24 bits)

Shared key (64 or 128 bit)

Plaintext

Cipher stream

XOR

Figure 1

The final packet can be denoted as , where *v* is the IV, *M* is the message, is the checksum of *M*, and *k* is the shared key. The packet is received and decrypted by XORing the ciphertext over the (keystream, giving the original plaintext. This is written as . Because XOR is associative, the parentheses can be removed: .[[12]](#footnote-13) Using this same property, parentheses can be placed around the two keystreams: . It can then be determined that is an array of 0s, which will henceforth be denoted , because . , thus, the original plaintext is found as . The message is then extracted from the plaintext and it's checksum is found. If the checksum in the plaintext matches the calculated plaintext, then the message is determined to be properly transmitted.[[13]](#footnote-14) However, the first signs of failure immediately become evident. The key is shared between the client and router; which means that if a third party finds the key, then any packet transmitted either by the client or router can be deciphered by the third party. As well, if we have a message M1 and M2, then

Thus, the XOR of two ciphertexts is the XOR of two plaintexts, because the keystreams cancel out to an array of 0s.[[14]](#footnote-15) This is dangerous because if one of the plaintexts is known, suppose M1, is known, then will be known, thus M1,c(M1)⊕M2,c(M2) can be XORed with , yielding

Thus, if one packet is known, it can be used to calculate the contents of other packets. However, this is based on the assumption that the IV, which is labeled *v* in the above equations, is constant for both packets.

This property can be extended further, by using one of the ciphertexts, where the plaintext, , is known, then we can find the keystream by

Thus, an attack could exist which takes the cipherstream (the output) of an RC4 cipher and yields the key (the input, ). Should such an attack exist, then deprecation of WEP may not be enough, given the ease at which colliding IVs can be found.

These properties open up the possibility for several attacks. Should one have known plaintexts and ciphertexts, such that the ciphertext is the encrypted plaintext, for each possible value for *v*, packets could be XORed over the appropriate plaintext (the one with the same IV) to obtain the plaintext; however, this would take a significant quantity of space, given that there are 224 IV values, or 16777216 values. Thus, two issues with this decryption method arise: a need to know the plaintext of each ciphertext, which requires injecting traffic in some fashion and a need to have large maps of IV to plaintexts. Such a weakness exists because IVs are given freely, unencrypted, as shown in figure 1. Supposing that unique IVs were generated at an average of 1 second per IV, which is somewhat frequent, it would take days to gather all the needed IVs to be able to decrypt all packets.This attack can be improved, not in time but in space efficiency, by obtaining the cipher streams for each IV. This will take the same amount of time and use less space. Because 194 days is long enough that only a dedicated attacker would purse the attack, this attack does not warrant the removal of WEP.

The most straightforward way of obtaining the key is through brute force. The IV is given, so in a 64 bit WEP encryption only 40 bits are hidden. Thus, the number of possible keys is ; however, for a 50% chance of finding the key, only of the keys need to be tested, or , or 549,755,813,888. This is not unreasonable, especially with supercomputer access. Given enough time and a fast enough processor, the key can be found. However, this method, found in appendix A, can take several months on a standard processor.[[15]](#footnote-17)This does not validate the immediate removal of WEP, given that several months of computing time is impractical for those who cannot dedicate a whole computer’s resources for months at a time uninterrupted. As well, an option to solve this issue is to switch to 128 bit WEP, hiding 104 bits, increasing the total keys to , with 50% success at . At times more key combinations than 64 bit WEP, 128 bit WEP thwarts any intruder from attempting a brute force on a wireless network. As such, it seems that 128 WEP may not only be worth leaving as a valid option, but also worth reversing the deprecation, assuming that brute force is the only feasible option.

Although brute force is the most straightforward way of cracking WEP, there exist statistical attacks on the RC4 cipher stream which will provide the input used to create the stream. Certain inputs have a strong correlation with specific bits, such that only a handful of bits are actually used to compute the stream. Thus, the task of discovering the key becomes far simpler. Once a cipher that fulfills the requirements above is obtained by the recover method shown above, the attack on the cipher can be used to determine the key.

Because the key is so easy to derive – some applications, such as AirCrack-ng, can retrieve a 64 bit key in less than 5 minutes[[16]](#footnote-18) – WEP is clearly not secure. A standard user, however, is unlikely to beaware of the difference between WEP and alternatives, such as WPA. Also, some older network cards do not support WPA, which is probably why WEP is still offered. On such network cards, two alternative configurations exist: MAC filter and open. Open networks are clearly not an option, because they allow anyone to join the network, and thus anyone can inject and intercept traffic. MAC filters are only slightly more secure, preventing traffic injection; but, they do not encrypt or protect traffic in any way. Also, it is trivial to exploit the fact that MAC addresses are stored, unencrypted, in packets. To obtain access to a MAC filter protected network, one can simply intercept a packet, observe the MAC address, and change their local MAC address (temporarily or permanently) to that address. Though this method is fairly simple, recovering a WEP shared key is also fairly simple with tools such as AirCrack-ng. Using WEP will delay an intruder for approximately 5 minutes, relative to an open network. A MAC filter will only delay an intruder for a few milliseconds, assuming the intrusion process is automated. As well, a MAC filter does not prevent interception of packets containing valuable information such as credit cards and social security numbers. Thus, WEP is preferable to a MAC filter. However, if the WEP key is not changed every 5 minutes, the network is vulnerable. Thus, a modified WEP encryption with a protocol for key modification would suffice to make WEP protective. For example, special packets could be transmitted from the router to the client every 5 minutes that specify the new key. However, if network cards are to be updated with the software to use this modified WEP, they may as well be updated for WPA. It can be concluded that WEP is only useful for network cards that were created before the release of WPA and cannot be updated.There is no doubt that an insecure network is dangerous.

If the evidence provided above is not enough to prove so, a real world case of wireless security intrusion will be more than enough. In 2007, long after the deprecation of WEP, a security breach at TJ Maxx cost almost 4.5 billion dollars in stolen credit cards. Intruders broke into their wireless network and stole 94 million credit cards.[[17]](#footnote-19) A retail store was using WEP as their network encryption. An attacker noticed this and gained access to their network, providing access to all computers on the network.[[18]](#footnote-20)After the attacker had successfully accessed one of TJ Maxx’s stores, the attacker decided to take the attack the TJ Maxx’s headquarters, where he proceeded to collect millions of credit card numbers.[[19]](#footnote-21) The attacker had access to TJ Maxx’s credit card database, which had lacked encryption and had far too much information.[[20]](#footnote-22) Had any one of these 3 problems been fixed, TJ Maxx would not have had the same scale of problems. Had TJ Maxx simply switched to WPA, $4.5 billion would not have been stolen. Doing so would not have been very costly – at most, wireless routers would have to be updated, and routers can go for as cheap as 50 USD per router.[[21]](#footnote-23) If the appropriate wireless routers are in place, a few buttons would have to be pressed to switch the encryption from WEP to WPA. Even if TJ Maxx had encrypted their files, any sensitive details that had been transmitted over their wireless network would have been stolen. Had the attacker been wary of detection or unable to access the files, they could simply passively collect packets and decrypt them, giving access to every piece of data transmitted via the local area network. Using WEP is clearly not on option if anything important is being transmitted or is located on a networked computer. Perhaps the scariest part of this attack is that the skill needed to perform such an attack is minimal – the software and hardware for gaining access to and reading packets from a WEP encrypted network is all available online.

Five years ago, the backwards compatibility argument for WEP would have been valid. However, WEP was deprecated 10 years ago. Given that computers are meant to be replaced every 3 years, computers that are over 10 years of age should have been replaced threefold and are due for an update. Given the current price of computers, most people who can afford an internet connection and wireless router can also afford a new computer. Thus, there should be no backwards compatibility issue with removing WEP from wireless routers in advanced computing environments. As such, no network card or router should be incompatible with WPA, and thus WEP should be unnecessary and removed. The mere existence of the option to use WEP should not be offered for the same reason low entropy passwords should not be allowed – those who don’t understand the necessity for security will tend to stick with the easiest thing, which may be WEP.[[22]](#footnote-24) Thus, WEP should be removed from all IEEE 802.11 complying routers in order to maximize security and minimize crime, such as the catastrophe that occurred when security was breached at TJ Maxx.

Appendix A

This appendix has a brute force method for searching for matches in keys. To search for a specific key with the plaintext and ciphertext, one would set **short**[] s to the ciphertext and sample to the plaintext. The while loop would no longer be necessary as only one ciphertext is being examined, rather than all ciphertexts for one plaintext. As it is set up, this program would count the number of keys for which the key can be identified for a sample text of length 10. Calling run() will yield the calculation after an extended period of time on a powerful computer.

Note:

It is likely not of interest to read the entire source. While it is only less than 300 lines, it is also very condensed. For example, it should suffice to understand that the KeyFactory and KeyNode classes work together to create every possible key for brute force purposes.

**package** rc4;

/\*\*

\* Tests the number of matches

\* **@author** Ryan Amos

\*/

**public** **class** MatchTest

{

**private** **static** **final** **int** maxThreads = 100;

**private** **static** String sample = "helloworld"; //10 characters

**private** **long** numMatches = 0; //to keep track of everything

**private** KeyFactory factory = **new** KeyFactory(); //main KeyFactory

**private** **int** numFinished = 0;

**public** MatchTest()

{

}

**public** **long** run()

{

**while**(factory.hasNext()){

**int**[] key = factory.next();

**short**[] s = getRC4(key);

/\* initialize the KeyFactory to run in this thread \*/

KeyFactory f = factory.clone();

f.next();

/\* initialize the other threads \*/

**for**(**int** i = 0; i < *maxThreads*; i++)

{

startThread();

}

/\* run this thread \*/

**while**(f.hasNext())

{

**if**(s.equals(getRC4(f.next())))

numMatches++;

}

/\* wait for all other threads to finish so that saving is synchronized \*/

**while**(numFinished != *maxThreads*)

{

**try**

{

Thread.*sleep*(1000);

} **catch** (InterruptedException e)

{

e.printStackTrace();

}

}

numFinished = 0;

}

**return** numMatches;

}

**private** **void** startThread()

{

**final** **short**[] set = getRC4(factory.next()); //to compare to

**final** KeyFactory fact = factory.clone(); //generator our own key factory

**new** Thread(**new** Runnable()

{

@Override

**public** **void** run()

{

**while**(fact.hasNext())

{

**if**(set.equals(getRC4(fact.next()))) //compare

numMatches++;

}

numFinished++;

}

}).start();

}

**private** **short**[] getRC4(**int**[] key){

**short**[] vals = **new** **short**[10];

RC4 rc4 = **new** RC4(key);

**int** i = 0;

**for** (**char** c : sample.toCharArray())

{

vals[i] = (**short**)rc4.rc4((**int**)c);

i++;

}

**return** vals;

}

}

**package** rc4;

/\*\*

\* RC4 Algorithm source: http://web.archive.org/web/20080404222417/http://cypherpunks.venona.com/date/1994/09/msg00304.html

\* **@author** Ryan Amos

\*/

**public** **class** RC4

{

**public** **int**[] state;

**public** **int** x;

**public** **int** y;

**public** RC4(**int**[] key)

{

**int** index1 = 0;

**int** index2 = 0;

x = 0;

y = 0;

state = **new** **int**[256];

**for** (**int** i = 0; i < state.length; i++)

state[i] = i;

**for** (**int** i = 0; i < state.length; i++)

{

index2 = (key[index1] + state[i] + index2) % 256;

**int** temp = state[index1];

state[index1] = state[index2];

state[index2] = temp;

index1 = (index1 + 1) % key.length;

}

}

**public** **int** rc4(**int** b)

{

x = (x + 1) % 256;

y = (state[x] + y) % 256;

**int** temp = state[x];

state[x] = state[y];

state[y] = temp;

**int** xorIdx = (state[x] + state[y]) % 256;

**return** b ^ state[xorIdx];

}

}

**package** rc4;

/\*\*

\* Generates keys using a brute force method

\* **@author** Ryan Amos

\*/

**public** **class** KeyFactory **implements** java.util.Iterator<**int**[]>, Cloneable{

**private** KeyNode root;

**public** KeyFactory(){

root = **new** KeyNode(0, 8, 0, **null**);

}

**private** KeyFactory(KeyNode root){

**this**.root = root;

}

@Override

**public** **boolean** hasNext() {

**return** root.value < 256; //root has extended to maximum value

}

@Override

**public** **synchronized** **int**[] next() {

**if**(!hasNext())

**return** **null**;

**int**[] ret = **new** **int**[root.maxDepth]; //key to return

KeyNode node = root;

//get this key

**for** (**int** i = 0; i < ret.length; i++) {

ret[i] = node.value;

**if**(i != ret.length - 1)

node = node.focus;

}

//Advance to the next key

**if**(node.parent.shiftFocus() == **null**)

root = **new** KeyNode(0, 8, root.value + 1, **null**);

**return** ret;

}

@Override

**public** **void** remove() {

**throw** **new** UnsupportedOperationException();

}

@Override

**public** KeyFactory clone(){

KeyNode n = **new** KeyNode(root.depth, root.maxDepth, root.value, **null**);

KeyNode r = root;

KeyNode z = n; //pointer for later access

**while**(n.focus != **null**){

n.focus = **new** KeyNode(r.focus.depth, root.maxDepth, r.focus.value, n);

r = r.focus;

n = n.focus;

}

**return** **new** KeyFactory(z);

}

}

**package** rc4;

/\*\*

\* Represents a part of a key (one byte of the key array).

\* Children are the subsequent bytes and parents are the preceding bytes.

\* **@author** Ryan Amos

\*/

**public** **class** KeyNode **implements** Cloneable{

**public** **int** depth;

**public** **int** maxDepth;

**public** **int** value;

**public** KeyNode parent;

**public** KeyNode focus;

**public** KeyNode(**int** depth, **int** maxDepth, **int** value, KeyNode parent) {

**this**.depth = depth;

**this**.maxDepth = maxDepth;

**this**.value = value;

**this**.parent = parent;

//System.out.println(depth + " : " + maxDepth);

**if**(depth == maxDepth - 1)

focus = **null**;

**else**

focus = **new** KeyNode(depth + 1, maxDepth, 0, **this**);

}

/\*\*

\* Advances the node. If the node cannot be advanced, advances the parent node

\* **@return** The node that was advanced

\*/

**public** KeyNode shiftFocus(){

**int** nextVal = focus.value + 1;

**if**(nextVal < 256 && depth < maxDepth){

focus = **new** KeyNode(depth + 1, maxDepth, nextVal, **this**);

**return** focus;

} **else** **if**(parent != **null**){

**return** parent.shiftFocus();

} **else** {

**return** **null**;

}

}

}

1. Nikita Borisov, Ian Goldberg, and David Wagner, “Intercepting Mobile Communications: The Insecurity of 802.11” (paper published in the proceedings of the Seventh Annual International Conference on Mobile Computing and Networking, July 16-21, 2001). http://www.isaac.cs.berkeley.edu/isaac/mobicom.pdf. (accessed April 27, 2011). [↑](#footnote-ref-2)
2. David Wagner. “Wireless Security.”<http://www.fcc.gov/realaudio/presentations/2002/042902/wagner.pdf>. Accessed 30 April, 2011. [↑](#footnote-ref-3)
3. <http://www.zdnet.com/blog/ou/retailers-havent-learned-from-tjx-still-running-wep/487> [↑](#footnote-ref-4)
4. I observed this by examining routers from several companies. Encryption always defaulted to either none (unsecured), or WEP, even when WPA was available. [↑](#footnote-ref-5)
5. A MAC address can be changed using an application such as macchanger: <http://www.alobbs.com/macchanger/> [↑](#footnote-ref-6)
6. Borisov, Goldberg, and Wagner [↑](#footnote-ref-7)
7. Ibid. [↑](#footnote-ref-8)
8. Ibid. [↑](#footnote-ref-9)
9. Germundsson, Roger and Weisstein, Eric W. "XOR."From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/XOR.html [↑](#footnote-ref-10)
10. <nobody@jpunix.com>, "RC4 Source Code," Private e-mail message to <cypherpunks@toad.com>. 9 September 1994. <http://web.archive.org/web/20080404222417/http://cypherpunks.venona.com/date/1994/09/msg00304.html>. Accessed 7 January, 2012. [↑](#footnote-ref-11)
11. Borisov, Goldberg, and Wagner. [↑](#footnote-ref-12)
12. Germundsson, Roger and Weisstein. [↑](#footnote-ref-13)
13. Borisov, Goldberg, and Wagner. [↑](#footnote-ref-14)
14. Nikita Borisov, Ian Goldberg, and David Wagner. “(In)Security of the WEP Algorithm.” ISAAC Group Home Page.Web. 02 Dec. 2010. <http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html>. [↑](#footnote-ref-15)
15. On my computer, a 2.5GHz dual core, I ran the program listed in appendix B. It was finishing around .001% per minute, or 1% per 1000 minutes. Given that for 100% success, 100% must be achieved, it can be deduced that for 100% success, minutes are needed. This is days, or months. [↑](#footnote-ref-17)
16. I performed cracking tests on my wireless router using AirCrack-ng. I maximized the communication through large downloads and attempted to decrypt every minute. Some of the tests took less than 5 minutes. [↑](#footnote-ref-18)
17. Berg, Gerry G., Freeman, Michelle S., Schneider, Kent N. "Analyszing the TJ Maxx Data Security Fiasco."The CPA Journal.< http://www.nysscpa.org/cpajournal/2008/808/essentials/p34.htm> (accessed 9/11/11). [↑](#footnote-ref-19)
18. Ibid. [↑](#footnote-ref-20)
19. Ibid. [↑](#footnote-ref-21)
20. Ibid. [↑](#footnote-ref-22)
21. An example wireless router for $50 can be found at <http://www.amazon.com/D-Link-DIR-601-Wireless-N-Home-Router/dp/B002VJL0OS>. Accessed 6 January, 2012. [↑](#footnote-ref-23)
22. WEP can be found as the default security option on some wireless routers [↑](#footnote-ref-24)